```
/***********************************************************************

            Filename: rule.pc

            Description: This file contains the main() function for
                         the local-rule implementation.

***********************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include "message.h"
#include "stage2.h"
#include "../mdbms/mdbms.h"

EXEC SQL INCLUDE sqlca;
EXEC SQL BEGIN DECLARE SECTION;
    VARCHAR username[20];
    VARCHAR password[20];
EXEC SQL END DECLARE SECTION;

#define  ERROR(str)  {printf("\n\n%s\n",str);exit(1);}

extern tIagg **Iagg;
extern char  **runtime_item;

tBase       **Ibase;
tFinal      **Iagglist;
rtrigger    *rule_trigger=NULL;
int         trigger_count=0;
tnode       *rule_tree = NULL;
char        *rule_id = NULL;
tIoepr      **Ioepr;


main(int argc, char *argv[])
{
    int     i, j;
    char    command[256];

    FILE *ftxt;         /* for storing the rule text */
    FILE *ftrg;         /* for storing the trigger information */

    tnode   *a_rule=NULL, **rule_list=NULL;

    strcpy(username.arr, "MDB");
    username.len = strlen(username.arr);
    strcpy(password.arr, "INFOSYST");
    password.len = strlen(password.arr);
    EXEC SQL WHENEVER SQLERROR GOTO sqlerror;
    EXEC SQL CONNECT :username IDENTIFIED BY :password;

    ftxt = fopen("ruletext.doc", "r");
    ftrg = fopen("trig_mes.doc", "w");


    if (!ftxt)
       ERROR("main: Ruletext file doesn't exist !!")

    rule_trigger = NULL;
    rule_list = (tnode **) new_list();


    i = 0;
    while ((a_rule = get_rule_tree(ftxt)) != NULL)
        {

            rule_tree = a_rule;
            rule_id = (char *)malloc(strlen(rule_tree->nvalue)+1);
            strcpy(rule_id, rule_tree->nvalue);


            /* store the trigger information into trig_mes.doc */

            display_trigger_message(ftrg, rule_trigger);

            /* determine data items to be retrieved and updated, etc */

            prepare_data_set();


            /* generate the MQL views and final factbase */

            gen_mql_view();


            /* generate the local DBMS views and final factbase */

            gen_local_view();   /* original gqs */


            /* generate the local DBMS code for rule execution */

            rule_codegen();  /* in rulegen.c */


            trigger_count = 0;
            rule_trigger = NULL;
            free(Ibase);

            rule_list = (tnode **) add_elem(i, (tptr*) rule_list, (tptr)
a_rule);
            i++;
       }

    fclose(ftrg);
    fclose(ftxt);
    sprintf(command, "cp trig_mes.doc ../timer/");
    system(command);

    EXEC SQL ROLLBACK;
    return 0;
sqlerror:
    printf("\n\n% .70s \n\nin file %s at line %d\n",
sqlca.sqlerrm.sqlerrmc,__FILE__,__LINE__);
    EXEC SQL WHENEVER SQLERROR CONTINUE;
    EXEC SQL ROLLBACK;
    exit(1);
}
```

```
/**************************************************************************
            Filename: factbase.c

            Description: Determine which data items in the rule
                         will be in the final factbase and build the
                         global structure for final factbase for later
                         processing.

**************************************************************************/

#include <stdio.h>
#include "message.h"
#include "stage2.h"
#define    NULLKEY        "                    "

#define  ERROR(str)  {printf("\n\n%s\n",str);exit(1);}

extern tnode      *rule_tree;
extern tItab      **Itab;
extern tIoepr     **Ioepr;
extern tIagg      **Iagg;
extern tIset      **Iset;
extern char       **runtime_item;
extern tBase      **Ibase;    /* the final factbase items from main views*/
extern tFinal     **Iagglist;  /* the final factbase items from aggregate
views */


/**************************************************************************
prepare_data_set()

Descritption:     Determine queries and prepare the sets of data items for
                  the factbase creation.
**************************************************************************/

prepare_data_set()
{
      int agg_count = 0;

      Ibase = NULL;
      Iagg = NULL;
      Iagglist = NULL;
      runtime_item = (char **)new_list();
      Ibase = (tBase **)new_list();

      identify_query();  /* indentify the independent queries */

      build_set();  /* build sets for later use,such as Ioepr, Itab,etc*/

      complete_set();   /* complete those sets */

      build_Iagg(rule_tree, &agg_count);  /* build aggregation info. */

      build_final();  /* build the final factbase */
}


/**************************************************************************
build_final()

Description:
            Build the final factbase and store them into Ibase and Iagglist.
**************************************************************************/

build_final()
{
      int i;

      int k;

      /* check the rule-tree to mark which item be final in the Itab list */
      i = 0;
      while ((rule_tree->children)[i] != NULL)
            {
                determine_final_items((rule_tree->children)[i]);
                ++i;
            }


      build_final_from_Itab();

      build_final_from_Iagg();

      Add_updated_oepr_key_into_final();

      /* test */
      k=0;
      printf("\n\nThe end of the build_final: THE FINAL ITEMS \n");

      while ((Ibase[0]->itemlist)[k] != NULL)
            {
                printf("TEST: %s, %s, %d\n", (Ibase[0]->itemlist)[k]->
itemname, (Ibase[0]-> itemlist)[k]->iformat, (Ibase[0]->itemlist)[k]->
ilength);
                ++k;
            }

}


/**************************************************************************
int card(tptr *list)

tptr *list
                  The list being calculated.
Description:
            Calculate the total number of elements in the list.
**************************************************************************/
```

```
int card(tptr *list)
{
     int i = 0;

     while (list[i] != NULL)
            ++i;

     return(i);
}


/**************************************************************************
build_final_from_Itab()

Description:
            Select the items in Itab with is_final=TRUE and put them into
            final list.
**************************************************************************/

build_final_from_Itab()
{
      tItab **list;
      int i, k, count;

      tFinal *item;
      tBase  *base;


      /* Get the items being marked in Itab and put into the Final list for
each main views (alias) */

      for (list = Itab; *list != NULL; list++)
      {
            if ((*list)->is_final == TRUE)
            {

                  /* build the tFinal item element */


                  item = (tFinal *)malloc(sizeof(tFinal));

                  CopyStr(item->itemname, (*list)->itemname);

                  CopyStr(item->oeprname, (*list)->oeprname);

                  printf("Test: %s\n", (*list)->applname);

                  get_datatype_from_mdb (&item, (*list)->applname);


                k = 0;
                while ((Ibase[k] != NULL)&&(strcmp(Ibase[k]->viewname,
(*list)-> viewname)))
                  ++k;
                if (Ibase[k] == NULL)  /* related to the k alias */
                {
                  base = (tBase *)malloc(sizeof(tBase));

                  base->viewname = (*list)->viewname;


                  base->itemlist = (tFinal **)new_list();

                  base->itemlist = (tFinal **)add_elem(0, (tptr *)(base-
>itemlist), (tptr)item);


                  Ibase = (tBase **)add_elem(k, (tptr *)Ibase, (tptr)base);
               }
               else
               {
                  Ibase[k]->itemlist = (tFinal **)add_elem(card(Ibase[k]->
itemlist), (tptr *)(Ibase[k]->itemlist), (tptr)item);
               }

            }
      }

}


/***************************************************************************
Add_updated_oepr_key_into_final()

Description:
            For the updated items, we need to add their primary key items
            into the final list.
***************************************************************************/

Add_updated_oepr_key_into_final()
{
    tItab **list;
    int i,j,k,m;
    tFinal *item;

    for(list = Itab; *list != NULL; list++)
        {
      if ((*list)->status == ITEM_U)  /* check only the updated item */
      {
            /* find the oepr of the updated item */
            i = 0;
            while ((Ioepr[i] != NULL)&&(strcmp(Ioepr[i]->oeprname, (*list)->
oeprname)))
                  ++i;

            if (Ioepr[i] == NULL)
                  ERROR("we can not find the oepr from Ioepr!\n")
            /* find which Ibase this oepr belong to */
            k = 0;
            while ((Ibase[k] != NULL)&&(strcmp(Ibase[k]->viewname, (*list)-
>viewname)))
                  ++k;

            if (Ibase[k] == NULL)
                  ERROR("no viewname in Ibase\n")


            /* Add uniquely this key items into this Ibase */
            j = 0;
            while ((Ioepr[i]->key_of_oepr)[j] != NULL)
            {
                  /* check whether the key item is already in the Ibase */

                  m = 0;
                  while (((Ibase[k]->itemlist)[m] != NULL)&&
                         (strcmp((Ibase[k]->itemlist)[m]->itemname,
(Ioepr[i]-> key_of_oepr)[j])||strcmp((Ibase[k]->itemlist)[m]->oeprname,
Ioepr[i]->oeprname)))
                        ++m;

                  if ((Ibase[k]->itemlist)[m] == NULL)
                  {
                        item = (tFinal *)malloc(sizeof(tFinal));

                        CopyStr(item->itemname, (Ioepr[i]->key_of_oepr)[j]);
                        CopyStr(item->oeprname, Ioepr[i]->oeprname);

                        get_datatype_from_mdb (&item, (*list)->applname);

                        Ibase[k]->itemlist = (tFinal **)add_elem(m,
(tptr*)(Ibase[k]-> itemlist), (tptr)item);
                  }

                  ++j;
            }

      } /* if ITEM_U */

   }/* for loop */

}


/**************************************************************************
build_final_from_Iagg()

Description:
            Determine the items of the aggregate functions in the rule
            which will be put in the final factbase.

**************************************************************************/
```

```
build_final_from_Iagg()
{
    int i;
    tFinal *item;


    Iagglist = (tFinal **)new_list();

    if(Iagg == NULL) return;

    for(i = 0; Iagg[i] != NULL; i++)
        {

            item = (tFinal *)malloc(sizeof(tFinal));

            if(strcmp(Iagg[i]->fk, "exist") && strcmp(Iagg[i]->fk, "count"))
            {

            CopyStr(item->itemname, Iagg[i]->dk);
            CopyStr(item->oeprname, Iagg[i]->oeprname);

            get_datatype_from_mdb(&item, Iagg[i]->applname);

            free(item->itemname);

                  item->itemname = (char *) malloc(strlen(Iagg[i]->fk)+
strlen(Iagg[i]->dk)+5);
                  sprintf(item->itemname, "%s%d_%s\0", Iagg[i]->fk, i,
Iagg[i]->dk);

      }
      else
      {
            item->itemname = (char *)malloc(strlen(Iagg[i]->fk)+4);
            sprintf(item->itemname, "%s%d\0", Iagg[i]->fk, i);
            item->oeprname = NULL;
            item->iformat = "INTEGER";
      }

      Iagglist = (tFinal **)add_elem(i, (tptr *)Iagglist, (tptr)item);

     }

}


/***************************************************************************
determine_final_items(node)

tnode *node
            The node of the rule tree.
Description:
            Traverse the rule-tree to determine which item in the Itab list
            will be in the final factbase.
```

```
**************************************************************************/

determine_final_items(node)
tnode *node;
{
    tItab **list2;
    int i, flag;

    if (node->ntype == N_IF)
       {
        i = 0;
        while ((node->children)[i] != NULL)
        {
                determine_final_items((node->children)[i]);
                ++i;
        }

        return;
       }

   if(node->ntype == N_UPDATE_DIR)
        {
        return;
        }

    if(node->ntype == N_FUNCT &&
        (!strcmp((char*)(node->nvalue), "exist") ||
         !strcmp((char*)(node->nvalue), "count") ||
         !strcmp((char*)(node->nvalue), "sum") ||
         !strcmp((char*)(node->nvalue), "max") ||
         !strcmp((char*)(node->nvalue), "min") ||
         !strcmp((char*)(node->nvalue), "avg")))
        {
      return;
        }

    if(node->ntype == N_FUNCT)
        {
        for(i=0; node->children[i] != NULL; i++)
            {
            determine_final_items(node->children[i]);
            }
        return;
        }

    if(node->valuetype == NOT)
        {
        determine_final_items(node->children[0]);
        return;
        }

    if(node->ntype == N_OPER)  /* for operation like AND, OR, *, + ,-, etc */
        {
        determine_final_items(node->children[0]);
        determine_final_items(node->children[1]);

        return;
        }

    if(node->ntype == N_ITEM)
        {
        for(list2 = Itab; *list2 != NULL; list2++)
        {
            if(!strcmp((*list2)->applname, node->applname) &&
               !strcmp((*list2)->oeprname, node->oepr) &&
               !strcmp((*list2)->itemname, (char *)(node->nvalue)))
            {
                  (*list2)->is_final = TRUE;
                   break;
          }
      }

        return;
        }

    if((node->valuetype == INTEGER)
       ||(node->valuetype == REAL)
       ||(node->valuetype == STRING))
    {
      return;
    }

}
```

```
/***********************************************************************

            Filename: local.pc

            Description: The interface to local system.

***********************************************************************/

#include <stdio.h>
#include "gqs.h"

extern char *rule_id;


/***********************************************************************
gen_local_view()

Description:
            Step 1: Read the MQL query form the input file
            Step 2: Process each view to find the physical data to
                    retrieve
            Step 3: generate code for local systems
***********************************************************************/

gen_local_view()
{

    int i,j,count=0, k;
    tview **views, **process_mql();
    tquery **queries=NULL;
    FILE *f1,*f,*trigger,*fsql;
    int nb_queries=0,sequence=0;
    char command[256], msg_id[11];
    char *fname_mql, *fname_sql;

    char **main_views=NULL;

    fname_mql = (char *)malloc(strlen(rule_id)+5);
    sprintf(fname_mql, "%s.mql", rule_id);

    fname_sql = (char *)malloc(strlen(rule_id)+5);
    sprintf(fname_sql, "%s.sql", rule_id);

    fsql = fopen(fname_sql, "w");

    /* Step 1: Read the MQL query form the input file */

    views = process_mql(fname_mql);


    /* Step 2: Process each view to find the physical data to retrieve */

    for (i=0;views[i]->viewname != NULL;i++)
    {
        qproc(views[i],views,&queries,&nb_queries,&sequence);
    }


    /* Step 3: generate code for local systems */

    main_views = (char **)new_list();

    for (i=0;i<nb_queries;i++)
    {

          /* Generate main( alias) views for each independent query */

          gen_sql_main_views(fsql, queries[i]);

          main_views = (char **)add_elem(i, (tptr *)main_views,
(tptr)queries[i]->viewname);
    }

    for (i=0;views[i]->viewname != NULL;i++)
    {

          /* Generate subviews related to agg-functions based on main views
*/
          if (!included(views[i]->viewname, main_views))
              gen_sql_agg_views(fsql,i,views[i],views,queries,nb_queries);

    }


    gen_sql_factbase(fsql);

    fclose(fsql);

    sprintf(command, "cp %s ../opsrule/", fname_sql);
    system(command);

}
```

```
/***********************************************************************

            Filename: getmdb.c

            Description: Get detailed information from Metadatabase.

***********************************************************************/

EXEC SQL INCLUDE sqlca;
EXEC SQL BEGIN DECLARE SECTION;
    VARCHAR username[20];
    VARCHAR password[20];
EXEC SQL END DECLARE SECTION;


#include <stdio.h>
#include "message.h"
#include "stage2.h"
#define        NULLKEY        "                    "


/***********************************************************************
get_datatype_from_mdb (item, applname)

tFinal **item
            The data item which is in the final factbase;
char *applname
            The application name;
Description:
            Given a data item in a application, find its data type and
            format form Metadatabase.
***********************************************************************/

get_datatype_from_mdb (item, applname)
tFinal **item;
char *applname;
{

    EXEC SQL BEGIN DECLARE SECTION;
        VARCHAR itemname[41];
        VARCHAR apname[21];
        VARCHAR I_IFORMAT[21];
        int     I_ILENGTH;
    EXEC SQL END DECLARE SECTION;

    strcpy(itemname.arr, (*item)->itemname);
    itemname.len = strlen(itemname.arr);

    strcpy(apname.arr, applname);
    apname.len = strlen(apname.arr);

    EXEC SQL WHENEVER SQLERROR GOTO sqlerror;

    EXEC SQL DECLARE item_cs CURSOR FOR
        SELECT I.IFORMAT, I.ILENGTH
```

```
            FROM MDB_ITEM I
            WHERE I.ITEMNAME = :itemname
            AND I.APPLNAME = :apname;
    EXEC SQL OPEN item_cs ;
    EXEC SQL WHENEVER NOT FOUND GOTO end_of_fetch;
    for ( ; ; )
        {
        EXEC SQL FETCH item_cs INTO :I_IFORMAT, :I_ILENGTH ;
        I_IFORMAT.arr[I_IFORMAT.len]='\0';

        CopyStr((*item)->iformat, I_IFORMAT.arr);

        (*item)->ilength = I_ILENGTH;

        }
end_of_fetch:
    EXEC SQL CLOSE item_cs ;
    return;
sqlerror:
    printf("\n\n% .70s \n\nin file %s at line %d\n",
sqlca.sqlerrm.sqlerrmc,__FILE__,__LINE__);
    EXEC SQL WHENEVER SQLERROR CONTINUE;
    EXEC SQL ROLLBACK;
    exit(1);
}

/***********************************************************************
get_detailed_funct (applname, funct_name, coded, loc, ret_type)

char  *applname
            The application name;
char  *funct_name
            The user-defined routine name;
char  **coded
            The program language this routine is coded;
char  **loc
            The location of this user-defined routine;
char  **ret_type
            The return type of the routine;
Description:
            For a given user-defined routine, find detailed information
            about this routine from Metadatabase. This function
            belongs to MDBMS system (Metadatabase  Management System)
      and re-used by local-rule implementation, so it'd not listed
            here.
***********************************************************************/
```

```
/***********************************************************************

            Filename: gen_mql.c

            Description: Generate the MQL queries for creation the local
                          factbase later.

***********************************************************************/

#include <stdio.h>
#include "message.h"
#include "stage2.h"

extern tnode      *rule_tree;
extern tItab      **Itab;
extern tIoepr     **Ioepr;
extern tIagg      **Iagg;
extern tIset      **Iset;
extern char       **runtime_item;
extern tBase      **Ibase;
extern char       *rule_id;


/***********************************************************************
gen_mql_view()

Description:      Generate the MQL qeuries for the local-rule.
***********************************************************************/

gen_mql_view()
{

      FILE *fmql;
      char *fname_mql;
      char command[256];


      fname_mql = (char *)malloc(strlen(rule_id)+5);
      sprintf(fname_mql, "%s.mql", rule_id);

      fmql = fopen(fname_mql, "w");

      gen_main_view(fmql); /* for independent queries */

      gen_agg_view1(fmql);  /* for sum, max, min, avg */

      gen_agg_view2(fmql);  /* for exists, count */

      gen_final_view(fmql);   /* the final factbase */

      fclose(fmql);

      sprintf(command, "cp %s ../opsrule/", fname_mql);
      system(command);
```

```
}


/***********************************************************************
gen_main_view(f)

FILE *f
            The *.mql file for storing MQL views.           ;
Description:
            For each independent query in the rule, generate a mql view.
            This function is the same function defined in ROPE stage2c.c
            and is re-used and modified for local-rule implementation.
***********************************************************************/

gen_main_view(f)
FILE *f;
{
    tIset **list;
    char  **list3;
    int i, count;

    for(list = Iset; *list != NULL; list++)
      {
      fprintf(f, "\nDefine view %s_%s", rule_id, (*list)->viewname);
      for(i = 0; (*list)->objlist[i] != NULL; i++)
          {
          count = Length((*list)->D_of_obj[i]);
          if(count != 0)
            fprintf(f, "\n        From OE/PR %s get", (*list)->objlist[i]);
          for(list3 = (*list)->D_of_obj[i]; *list3 != NULL; list3++)
            fprintf(f, "\n                %s", *list3);
          }
      fprintf(f,";\n");
      }
}


/***********************************************************************
gen_agg_view1(f)

FILE *f
            The *.mql file to store MQL views;
Description:
            Generate MQL views for SUM, MIN, MAX, AVG aggregate functions.
            This function is based on gen_2nd_view defined in ROPE stage2c.c
            and is re-used and modified for local-rule implementation.

***********************************************************************/

gen_agg_view1(f)
FILE *f;
{
    char  **list3;
    int i;
    if(Iagg == NULL)return;

    for(i = 0; Iagg[i] != NULL; i++)
        {

      /* for SUM, MAX, MIN, AVG, the SQL view name is : ruleid_sumK */

      if(strcmp(Iagg[i]->fk, "exist") && strcmp(Iagg[i]->fk, "count"))
          {
            fprintf(f, "\nDefine view %s_%s%d", rule_id, Iagg[i]->fk, i);

            fprintf(f, "\n    %s ( From view %s_%s get", Iagg[i]->fk,
rule_id, Iagg[i]->viewname);
            fprintf(f, "\n        %s", Iagg[i]->dk);

            for(list3 = Iagg[i]->Gk; *list3 != NULL; list3++)
                  fprintf(f, "\n       %s", (*list3));

            if(Iagg[i]->Sk != NULL)
                  fprintf(f, "\n    For %s );\n", Iagg[i]->Sk);
            else
                  fprintf(f, "\n    );\n");
         }
      }
}

/***********************************************************************
gen_agg_view2(f)

FILE *f
            The *.mql file for storing MQL views;
Description:
            Generate MQL views for EXISTS and COUNT aggrgate function.
            This function is based on gen_3rd_view() defined in ROPE
            stage2c.c and is re-used and modified for local-rule
            implementation.
***********************************************************************/

gen_agg_view2(f)  /* for EXISTS and COUNT */
FILE *f;
{
    char  **list3;
    int i;

    if(Iagg == NULL)return;

    for(i = 0; Iagg[i] != NULL; i++)
        {
        if(!strcmp(Iagg[i]->fk, "exist") || !strcmp(Iagg[i]->fk, "count"))
            {
            fprintf(f, "\nDefine view %s_%s%d", rule_id, Iagg[i]->fk, i);

            fprintf(f, "a");
            fprintf(f, "\n    From view %s_%s get", rule_id, Iagg[i]-
>viewname);
            for(list3 = Iagg[i]->Gk; *list3 != NULL; list3++)
                fprintf(f, "\n       %s", (*list3));
            fprintf(f, ";\n\n");
            fprintf(f, "Define view %s_%s%d", rule_id, Iagg[i]->fk, i);

            fprintf(f, "b");

          fprintf(f, "\n    %s ( From view %s_%s get", Iagg[i]->fk, rule_id,
Iagg[i]->viewname);
          for(list3 = Iagg[i]->Gk; *list3 != NULL; list3++)
            fprintf(f, "\n       %s", (*list3));
          fprintf(f, "\n    For %s );\n", Iagg[i]->Sk);

          fprintf(f, "\nDefine view %s_%s%d", rule_id, Iagg[i]->fk, i);

          fprintf(f, "\n    %s_%s%d", rule_id, Iagg[i]->fk, i);

          fprintf(f, "a\n        union ");

          fprintf(f, "%s_%s%d", rule_id, Iagg[i]->fk, i);

          fprintf(f, "b\n        on");
          for(list3 = Iagg[i]->Gk; *list3 != NULL; list3++)
                fprintf(f, "\n          %s", (*list3));
          fprintf(f, ";\n");
          }
        }
}


/**************************************************************************
gen_final_view(f)

FILE *f
            The *.mql file for storing MQL views;
Description:
            Generate the final factbase view for the rule execution.
            This final factbase generation for local-rule is different from
            the final view for global-rule in ROPE.
**************************************************************************/

gen_final_view(f)
FILE *f;
{
    tBase **list;
    char  **list3;
    tFinal **list4;
    int i, count;

    fprintf(f, "\nDistinct (");
    for(list = Ibase; *list != NULL; list++) /* for base-view alias */
        {
        fprintf(f, "\n    From view %s_%s get", rule_id, (*list)-> viewname);
        for(list4 = (*list)->itemlist; *list4 != NULL; list4++)
            fprintf(f, "\n        %s", (*list4)->itemname);
        }
    if(Iagg != NULL) /* for aggregation sub-view */
      {
        for(i = 0; Iagg[i] != NULL; i++)
          {
          fprintf(f, "\n    From view %s_%s%d", rule_id, Iagg[i]->fk, i);

          if(Iagg[i]->dk != NULL)  /* for sum,max,avg,min */
              {
                fprintf(f, " get\n        %s", Iagg[i]->dk);
              }
          else  /* for EXISTS and COUNT */
                fprintf(f, " get $$RESULT");
          }
        for(i = 0; Iagg[i] != NULL; i++)
            {
            count = Length(Iagg[i]->Gk);
            if(count != 0)
            {
              if(i==0)
                    fprintf(f, "\n    For");
              else
                    fprintf(f, "\n    And");
                for(list3 = Iagg[i]->Gk; *list3 != NULL; list3++)
                    fprintf(f, "\n        %s", (*list3));
              fprintf(f, " of view %s_%s =", rule_id, Iagg[i]->viewname);
                for(list3 = Iagg[i]->Gk; *list3 != NULL; list3++)
                    fprintf(f, "\n        %s", (*list3));
              fprintf(f, " of view %s_%s%d", rule_id, Iagg[i]->fk, i);
              }
          }
      }
    fprintf(f,"\n    );\n");
}
```

```
/***********************************************************************

            Filename: gen_sql.c

            Description: Generate the Orcale SQL views for the factbase
                         of the local-rule.

***********************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include "gqs.h"
#include "local.h"
#include "stage2.h"

#define EXIST 1
#define COUNT 2
#define SUM 3
#define AVG 4
#define MIN 5
#define MAX 6

extern char *rule_id;
extern tBase **Ibase;
extern tFinal **Iagglist;
extern tIagg **Iagg;

/***********************************************************************
gen_sql_factbase(f)

FILE* f           The *.sql file for storing the sql view creation;
Description:
            Generate the SQL view for the final factbase of the rule.
***********************************************************************/

gen_sql_factbase(f)
FILE *f;
{

      fprintf(f, "\ncreate view %s_factbase \n\t( ", rule_id); /* add
rule_id later */

      generate_final_itemlist(f);

      fprintf(f, " )\n");

      generate_final_select_from_clause(f);

      generate_final_where_clause(f);

}

/***********************************************************************
```

```
generate_final_itemlist(f)

FILE* f         The *.sql file for storing the sql view creation;
Description:
            The items appeared in the final view (including items from
            main views(Ibase) and aggregate views(Iagglist).
*********************************************************************/

generate_final_itemlist(f)
FILE* f;
{
      int i,k;
        int count = 0;

        i=0;
        while(Ibase[i] != NULL)
        {
                k = 0;
                while((Ibase[i]->itemlist)[k] != NULL)
                {
                 if (count == 0)
                    fprintf(f, "%s", (Ibase[i]->itemlist)[k]->itemname);
                 else
                    fprintf(f, ", %s", (Ibase[i]->itemlist)[k]->
itemname);

                 ++count;
                   ++k;
            }
            ++i;
      }


        k = 0;
        while (Iagglist[k] != NULL)
        {
            if (count == 0)
                 fprintf(f, "%s", Iagglist[k]->itemname);
            else
                 fprintf(f, ", %s", Iagglist[k]->itemname);
            ++k;
            ++count;
       }
}


/*********************************************************************
generate_final_select_from_clause(f)

FILE* f           The *.sql file for storing the sql view creation;
Description:
                  Generate the sql SELECT ... FROM ... clause.
*********************************************************************/

generate_final_select_from_clause(f)
```

```
FILE* f;
{

      int i,k;
      int count = 0;

      fprintf(f, "\tas select\n");

      i=0;
      while(Ibase[i] != NULL)
        {
                k = 0;
                while((Ibase[i]->itemlist)[k] != NULL)
                {
                  if (count == 0)
                        fprintf(f, "\t\t %s_%s.%s", rule_id, Ibase[i]->
viewname, (Ibase[i]->itemlist)[k]->itemname);
                  else
                        fprintf(f, ",\n\t\t %s_%s.%s", rule_id,
Ibase[i]->viewname, (Ibase[i]->itemlist)[k]->itemname);

                        ++k;
                        ++count;
                }
                ++i;
        }

        k = 0;

        while (Iagglist[k] != NULL)
        {
          if (count == 0)
          {
            if (Iagg[k]->dk != NULL)
                    fprintf(f, "\t\t %s_%s%d.%s_%s", rule_id, Iagg[k]->
fk, k, Iagg[k]->fk, Iagg[k]->dk);
            else
                fprintf(f, "\t\t %s_%s%d.%s_result", rule_id, Iagg[k]->
fk, k, Iagg[k]->fk, Iagg[k]->dk);
          }
          else
          {
            if (Iagg[k]->dk != NULL)
                    fprintf(f, ",\n\t\t %s_%s%d.%s_%s", rule_id,
Iagg[k]->fk, k, Iagg[k]->fk, Iagg[k]->dk);
                else
                    fprintf(f, ",\n\t\t %s_%s%d.%s_result", rule_id,
Iagg[k]->fk, k, Iagg[k]->fk, Iagg[k]->dk);
            }

            ++k;
            ++count;
        }
      count = 0;
        fprintf(f,"\n\t   from  ");

      k = 0;
      while (Ibase[k] != NULL)
      {
          if (count == 0)
            fprintf(f, "%s_%s", rule_id, Ibase[k]->viewname);
          else
            fprintf(f, ", %s_%s", rule_id, Ibase[k]->viewname);

            ++count;
            ++k;
      }

      k = 0;
      while (Iagg[k] != NULL)
      {
          if (count == 0)
            fprintf(f, "%s_%s%d", rule_id, Iagg[k]->fk, k);
          else
            fprintf(f, ", %s_%s%d", rule_id, Iagg[k]->fk, k);

            ++k;
            ++count;
      }

}


/***********************************************************************
generate_final_where_clause(f)

FILE* f           The *.sql file for storing the sql view creation;
Description:
                  Generate the sql WHERE ... clause.
***********************************************************************/

generate_final_where_clause(f)
FILE* f;
{
      int i, count;
      char** list;

      i = 0;
      while (Iagg[i] != NULL)
      {
          count = Length(Iagg[i]->Gk);
            if(count != 0)
                {
                if(i==0)
                    fprintf(f, "\n\t   where ");
                else
                    fprintf(f, "\n\t    and ");
                for(list = Iagg[i]->Gk; *list != NULL; list++)
```

```
                    fprintf(f, "%s_%s.%s = ", rule_id, Iagg[i]->
viewname, (*list));
                for(list = Iagg[i]->Gk; *list != NULL; list++)
                    fprintf(f, "%s_%s%d.%s", rule_id, Iagg[i]->fk, i,
(*list));

                }
        ++i;
      }

      fprintf(f, ";\n\n");

}


/***********************************************************************
gen_sql_main_views(f, query)

FILE *f
            The *.sql file for storing SQL view creation;
tquery *query
            The independent query related to each main views(alias);
Description:
            Generate the SQL views for each independent queries which
have
            included the additional join-conditions.
***********************************************************************/

gen_sql_main_views(f, query)
FILE *f;
tquery *query;
{

        int i, k, nb_selected = 0;
        char **filelst=NULL;
        int nb_file=0;
        tcond *cond;
        int first = 0;


        fprintf(f,"\ncreate view %s \n\t(", query->viewname);

        for (i=0; i < query->nb_items; i++)
        {
                if ((query->items)[i]->selected == 1)
                {
                        if (first != 0)
                                fprintf(f, ", %s", (query->items)[i]->
itemname);
                        else
                                fprintf(f, "%s", (query->items)[i]->
itemname);

                        ++nb_selected;
                  first = 1;
                }
        }

        fprintf(f, ")\n\tas select ");

        k = 0;
        for (i=0; i < query->nb_items; i++)
        {
                if ((query->items)[i]->selected == 1)
                {
                        fprintf(f, "%s.%s", (query->items)[i]->source->
file,
                                             (query->items)[i]->
itemname);
                        ++k;

                        if (k < nb_selected)
                           fprintf(f, ",\n\t\t ");
                }


        if (search_in_list((query->items)[i]->source-> file,filelst,
nb_file) == -1)

            {
            insert_in_list((query->items)[i]->source->file,&filelst,
nb_file);
            nb_file++;
            }
        }

        fprintf(f,"\n\tfrom  ");

        for (i=0;i<nb_file;i++)
        {
           if (i > 0)
               fprintf(f,", ");
           fprintf(f,"%s",filelst[i]);
        }

        for (i=0;(query->select != NULL) && (i<query->select->nb_cond);
i++)
        {
        cond = (query->select->conds)[i];
        if (i == 0)
            fprintf(f, "\n\twhere  ");
        else
            fprintf(f,"\n\t AND ");
        fprintf(f,"%s.%s ",cond->item1->source->file,cond->item1->
itemname);
        if (cond->valuetype != NULLSYM)
            fprintf(f,"%s ",rel_op("ORACLESQL",cond->operator));
        if (cond->valuetype == IDENTSYM)
            fprintf(f,"%s.%s ",cond->item2->source->file,cond->item2-
>itemname);
```

```
        else if (cond->valuetype == INTEGERSYM)
            fprintf(f,"%d",(int)(cond->value));
        else if (cond->valuetype == REALSYM)
            fprintf(f,"%g",*((double *)(cond->value)));
        else if (cond->valuetype == STRINGSYM)
            fprintf(f,"'%s'",(char *)(cond->value));
        else if (cond->valuetype == TRUESYM)
            fprintf(f,"TRUE");
        else if (cond->valuetype == FALSESYM)
            fprintf(f,"FALSE");
        else if (cond->valuetype == NULLSYM)
            {

            if (!strcmp(cond->operator,"="))
                fprintf(f,"IS ");
            else if (!strcmp(cond->operator,"<>"))
                fprintf(f,"IS NOT");
            else
                EXIT_GQS(62)
            fprintf(f,"NULL");
            }
        }
    fprintf(f,";\n");
    free_a_list(filelst);
}


/***********************************************************************
generate_view_itemlist(f, view, type)

FILE *f
            The *.sql file for storing SQL view creation;
tview *view
            The tview structure for storing aggregate view;
int type
            The type of aggregate function such as SUM, MAX, EXISTS,etc;
Description:
            Generate the items appear in the SQL view for the aggregate
            functions.
***********************************************************************/

generate_view_itemlist(f, view, type)
FILE *f;
tview *view;
int type;
{
      int i, first = 1;

      if (type == EXIST)
                fprintf(f, " exist_result");
        else if (type == COUNT)
                fprintf(f, " count_result");

      for (i=0;i<view->nb_items;i++)
      {
            if (((view->items[i]->itemname[0] != '$') ||
               (view->items[i]->itemname[1] == '$'))  &&
               (view->items[i]->source->oeprs[0][0] != '$'))
               {
                        if ((view->items)[i]->selected)
                        {
                        if (first)
                                {
                                        if (type == SUM)
                                                fprintf(f, "sum_%s",
(view->items)[i]->itemname);
                                        else if (type == AVG)
                                                fprintf(f, "avg_%s",
(view->items)[i]->itemname);
                                        else if (type == MIN)
                                                fprintf(f, "min_%s",
(view->items)[i]->itemname);
                                        else if (type == MAX)
                                                fprintf(f, "max_%s",
(view->items)[i]->itemname);
                              else
                                    fprintf(f, ", %s", (view->items)[i]-
>itemname);

                              first = 0;
                             }
                         else
                              fprintf(f, ", %s", (view->items)[i]->
itemname);

               }
      }
  }

}

/*********************************************************************
generate_SELECT_FROM_for_agg(f, view, flag, type)

FILE *f
            The *.sql file for storing SQL view creation;
tview *view
            The tview structure for aggregate function view;
int flag
            0 if EXIST is false, 1 if EXIST is true;
int type
            The type of aggregate function such as SUM, MAX, EXISTS,etc;
Description:
            Generate the SELECT...FROM... clause for EXIST and COUNT.
*********************************************************************/

generate_SELECT_FROM_for_agg(f, view, flag, type)
FILE *f;
tview *view;
int flag;
int type;
{
      int i;
      int first = 1;
      char **filelst=NULL;
        int nb_file=0;

      if (type == EXIST)
            fprintf(f, " %d ", flag);
      else if (type == COUNT)
           {
            if (flag == 1)
                  fprintf(f, " count(*) ");
            else if (flag == 0)
                  fprintf(f, " 0 ");
           }


      for (i=0;i<view->nb_items;i++)
        {
            if (((view->items[i]->itemname[0] != '$') ||
               (view->items[i]->itemname[1] == '$'))  &&
               (view->items[i]->source->oeprs[0][0] != '$'))
            {
                    if ((view->items)[i]->selected)
                    {

                        if (first)
                        {
                                        if (type == SUM)
                                                fprintf(f, "sum(%s)",
(view->items)[i]->itemname);
                                        else if (type == AVG)
                                                fprintf(f, "avg(%s)",
(view->items)[i]->itemname);
                                        else if (type == MIN)
                                                fprintf(f, "min(%s)",
(view->items)[i]->itemname);
                                        else if (type == MAX)
                                                fprintf(f, "max(%s)",
(view->items)[i]->itemname);
                              else
                                                fprintf(f, ", %s",
(view->items)[i]->itemname);

                                        first = 0;
                         }
                         else
                                fprintf(f, ", %s", (view->items)[i]->
itemname);

                    }

                    if (search_in_list((view->items)[i]->source->view,
filelst,nb_file) == -1)
```

```
                    {
                        insert_in_list((view->items)[i]->source->view,
&filelst,nb_file);
                        nb_file++;
                     }

            }

       }


       fprintf(f, "\n\t   from ");


        for (i=0;i<nb_file;i++)
        {
        if (i > 0)
            fprintf(f,", ");
        fprintf(f,"%s",filelst[i]);
        }

         free_a_list(filelst);

}


/***********************************************************************
generate_WHERE_clause(f, view, flag)

FILE *f
            The *.sql file for storing SQL views;
tview *view
            The view being processed;
int flag
            The flag used for EXISTS and COUNT;
Description:
            Generate the WHERE clause for aggregate functions.
***********************************************************************/

generate_WHERE_clause(f, view, flag)
FILE *f;
tview *view;
int flag;
{
      int i;
      tcond *cond;


      if (flag == 1)
          fprintf(f, "\n\t   where ");
      else if (flag == 0)
          fprintf(f, "\n\t   where NOT (");

        for (i=0;(view->select != NULL) && (i<view->select-
>nb_cond);i++)
```

```
        {
        cond = (view->select->conds)[i];
        if (i == 0)
            fprintf(f, " ");
        else
            fprintf(f," AND ");
        fprintf(f,"%s.%s ",cond->item1->source->view,cond->item1-
>itemname);
        if (cond->valuetype != NULLSYM)
            fprintf(f,"%s ",rel_op("ORACLESQL",cond->operator));
        if (cond->valuetype == IDENTSYM)
            fprintf(f,"%s.%s ",cond->item2->source->view,cond->item2-
>itemname);
        else if (cond->valuetype == INTEGERSYM)
            fprintf(f,"%d",(int)(cond->value));
        else if (cond->valuetype == REALSYM)
            fprintf(f,"%g",*((double *)(cond->value)));
        else if (cond->valuetype == STRINGSYM)
            fprintf(f,"'%s'",(char *)(cond->value));
        else if (cond->valuetype == TRUESYM)
            fprintf(f,"TRUE");
        else if (cond->valuetype == FALSESYM)
            fprintf(f,"FALSE");
        else if (cond->valuetype == NULLSYM)
            {
            if (!strcmp(cond->operator,"="))
                fprintf(f,"IS ");
            else if (!strcmp(cond->operator,"<>"))
                fprintf(f,"IS NOT");
            else
                EXIT_GQS(62)
            fprintf(f,"NULL");
            }
        }

      if (flag == 0)
            fprintf(f, " )");

}

/***********************************************************************
generate_GROUPBY_clause(f, view, type)

FILE *f
            The *.sql file for storing SQL views;
tview *view
            The view being processed;
int type
            The type of the aggregate functions;
Description:
            Generate the GROUP BY ... clause for the aggregate view.
***********************************************************************/

generate_GROUPBY_clause(f, view, type)
FILE *f;
```

```
tview *view;
int type;
{
      int i, first = 1;


      fprintf(f, "\n\t   group by ");

      for (i=0;i<view->nb_items;i++)
        {
            if (((view->items[i]->itemname[0] != '$') ||
                (view->items[i]->itemname[1] == '$'))  &&
                (view->items[i]->source->oeprs[0][0] != '$'))
            {
                if ((view->items)[i]->selected)
                {

                  if ((type == EXIST)||(type == COUNT))
                  {
                     if (first == 1)
                     {
                           fprintf(f, "%s.%s", (view->items)[i]->source-
>view,(view->items)[i]->itemname);

                           first = 0;
                     }
                     else
                          fprintf(f, ", %s.%s", (view->items)[i]-
>source->view,(view->items)[i]->itemname);
                  }
                  else
                  {
                     if (first == 1)
                        first = 2;
                     else if (first == 2)
                        {
                            fprintf(f, "%s.%s", (view->items)[i]->
source->view,(view->items)[i]->itemname);
                            first = 0;
                        }
                     else
                            fprintf(f, ", %s.%s", (view->items)[i]->
source->view,(view->items)[i]->itemname);
                  }


              }
          }
      }

      fprintf(f, ";\n");
}


/***********************************************************************
```

```
gen_sql_agg_views(f,i,view,views,queries,nb_queries)

FILE *f
            The *.sql file for storing the SQL views;
int i
            The index of which aggregate view will be processed;
tview *view
            The aggregate view to be processed;
tview **views
            The view list;
tquery **queries
            The independent queries list;
int nb_queries
            The number of independent queries;

Description:
            Generate SQL views for aggregate functions such as SUM, MAX,
            MIN, AVG, EXISTS, COUNT in the rule.
*************************************************************************/

gen_sql_agg_views(f,i,view,views,queries,nb_queries)
FILE *f;
int i;
tview *view,**views;
tquery **queries;
int nb_queries;
{

      tview *view1, *view2;
      int type;

      if (view->view1 != NULL)
      {

      fprintf(f,"\ncreate view %s \n",view->viewname);

      view1=find_view(view->view1,views);

      fprintf(f, "\tas select * \n\t   from %s\n\t ", view->view1);

      view2=find_view(view->view2,views);

      fprintf(f,"%s\n\t   select * \n\t   from %s;\n\n",view->
operator,view->view2);
      }

      if (view->function != NULL)
      {
            if (view->function->value_fct != NULL)
            {
                  if (!strcmp(view->function->value_fct, "EXIST"))
                  {

                        fprintf(f,"\ncreate view %s \n",view->viewname);
                        generate_view_for_function(f, view, 1, EXIST);
                        fprintf(f,"\ncreate view %s \n",views[i-1]->
viewname);
                        generate_view_for_function(f, view, 0, EXIST);
                  }
                  else if (!strcmp(view->function->value_fct, "COUNT"))
                  {
                        fprintf(f,"\ncreate view %s \n",view->viewname);
                        generate_view_for_function(f, view, 1, COUNT);

                        fprintf(f,"\ncreate view %s \n",views[i-1]->
viewname);

                        generate_view_for_function(f, view, 0, COUNT);
                  }
                  else
                  {
                         if (!strcmp(view->function->value_fct, "SUM"))
                              type = SUM;
                         else if (!strcmp(view->function->value_fct,
"AVG"))
                              type = AVG;
                        else if (!strcmp(view->function->value_fct,
"MIN"))
                              type = MIN;
                        else if (!strcmp(view->function->value_fct,
"MAX"))
                              type = MAX;

                        fprintf(f,"\ncreate view %s \n",view->viewname);

                        generate_view_for_function(f, view, 1, type);
                  }
            }
      }

}

/************************************************************************
generate_view_for_function(f, view, flag, type)

FILE *f
            The *.sql file for storing SQL views;
tview *view
            The view being processed;
int flag
            The flag used for EXISTS and COUNT;
int type
            The type of aggregate function;
Description
            The function is called by gen_sql_agg_views() for the
detailed
            generation of sql views for aggrgate functions:
************************************************************************/
```

```
generate_view_for_function(f, view, flag, type)
FILE *f;
tview *view;
int flag;
int type;
{

      fprintf(f, "\t( ");

      generate_view_itemlist(f, view, type);

      fprintf(f, " )\n");

      fprintf(f, "\tas select ");

      generate_SELECT_FROM_for_agg(f, view, flag, type);

      generate_WHERE_clause(f, view, flag);

      generate_GROUPBY_clause(f, view, type);

}
```

```
/***********************************************************************

            Filename: gen_pc.c

            Description: Generate the Oracle Pro*C code for the
                       local-rule execution.

***********************************************************************/

#include <stdio.h>
#include "message.h"
#include "stage2.h"
#define        NULLKEY        "                    "


extern tnode *rule_tree;
extern tItab **Itab;
extern tIoepr **Ioepr;
extern tIagg **Iagg;
extern tIset **Iset;
extern char  **runtime_item;
extern rtrigger  *rule_trigger;
extern tBase **Ibase;
extern tFinal **Iagglist;
extern char* rule_id;
int    Var_Num; /* count the number of host variables */


/***********************************************************************
rule_codegen()

Description:      Generate the Orcale Pro*C code for rule execution.
***********************************************************************/

rule_codegen()
{

      FILE *f;
      int i,first, use_flag, flag;
      char command[256];
      char *fname_pc;

      fname_pc = (char *)malloc(strlen(rule_id)+4);
      sprintf(fname_pc, "%s.pc", rule_id);

      f = fopen(fname_pc, "w");

      use_flag = 0; /* the use of routine */

      Var_Num = 0;

      include_header(f, rule_tree, &use_flag);

      main_function(f);
      fprintf(f, "\n%s()\n{\n\n", rule_id);

      host_variable_declaration(f);

      cursor_declaration(f);

      fetch_data_from_factbase(f);

      rule_execution(f);

      print_end(f);

      fclose(f);

      sprintf(command, "cp %s ../opsrule/", fname_pc);
      system(command);

}

/***********************************************************************
include_header(f, node, flag)

FILE *f
            The *.pc file for storing the Pro*C code of the rule;
tnode *node
            The node of the rule-tree;
int *flag
            For avoiding repeated include header file;
Description:
            Generate the header files to be included such as system
library
            and the file where the user-defined routines are stored.
***********************************************************************/

include_header(f, node, flag)
FILE *f;
tnode *node;
int *flag;
{
      fprintf(f, "#include <stdio.h>\n\n");

      include_user_defined_routine(f, node, flag);

      fprintf(f, "EXEC SQL INCLUDE sqlca;\n\n");

      fprintf(f, "EXEC SQL BEGIN DECLARE SECTION;\n");
      fprintf(f, "    VARCHAR username[20];\n");
      fprintf(f, "    VARCHAR password[20];\n");
      fprintf(f, "EXEC SQL END DECLARE SECTION;\n\n");

}
```

```
/***********************************************************************
main_function(f)

Description:      Generate the main() function for the rule execution.
***********************************************************************/

main_function(f)
FILE *f;
{
      fprintf(f, "main()\n{\n\n");

      fprintf(f, "    strcpy(username.arr, \"ops\");\n");
      fprintf(f, "    username.len = strlen(username.arr);\n");
      fprintf(f, "    strcpy(password.arr, \"ops\");\n");
      fprintf(f, "    password.len = strlen(password.arr);\n");
      fprintf(f, "    EXEC SQL WHENEVER SQLERROR GOTO sqlerror;\n");
      fprintf(f, "    EXEC SQL CONNECT :username IDENTIFIED BY
:password;\n\n");
      fprintf(f, "    %s();\n\n", rule_id);
      fprintf(f, "    return;\nsqlerror:\n");
      fprintf(f, "    printf(\"error in rule execution of %s\\n\");\n",
rule_id);
      fprintf(f, "    EXEC SQL WHENEVER SQLERROR CONTINUE;\n");
      fprintf(f, "    EXEC SQL ROLLBACK;\n");
      fprintf(f, "    exit(1);\n}\n\n");

}

/***********************************************************************
rule_execution(f)

Description:      Generate the Pro*C code for condition evaluation and
                  actions such as update, call user-routines, etc.
***********************************************************************/

rule_execution(f)
{

      int i,first, use_flag, flag;

      first = 1;

        flag = 0;

        i = 0;
        while ((rule_tree->children)[i] != NULL)
        {
        print_infix(f, (rule_tree->children)[i], &first); /* gen the
rule text in Pro*C code */

        ++i;
        }

      print_update_query(f); /* gen update SQL */
      fprintf(f, "\t}\n\n\t}\n"); /* for end of if(){} */
}


/*************************************************************************
print_end(f)

Description:      Generate the end section for a Pro*C function.
************************************************************************/

print_end(f)
FILE* f;
{
      fprintf(f, "end_of_fetch:\n");
      fprintf(f, "\tEXEC SQL CLOSE factbase;\n");
      fprintf(f, "\tEXEC SQL COMMIT WORK RELEASE;\n\treturn;\n\n");
      fprintf(f, "sqlerror:\n");
      fprintf(f, "\tEXEC SQL WHENEVER SQLERROR CONTINUE;\n\tEXEC SQL
ROLLBACK WORK;\n\texit(1);\n}\n");

}


/*************************************************************************
fetch_data_from_factbase(f)

Description:      Fetch the data items from the factbase of this rule
into the
            host-variables for rule execution.
************************************************************************/

fetch_data_from_factbase(f)
FILE *f;
{
      int i,k, count = 0;

      fprintf(f, "\tEXEC SQL OPEN factbase;\n\n");
      fprintf(f, "\tEXEC SQL WHENEVER NOT FOUND GOTO
end_of_fetch;\n\n");
      fprintf(f, "\tfor ( ; ; )\n");
      fprintf(f, "\t{\n\tEXEC SQL FETCH factbase INTO\n");

      i = 0;
      while (Ibase[i] != 0)
        {
                k = 0;
                while ((Ibase[i]->itemlist)[k] != NULL)
                {
                        if (count == Var_Num -1)
                        {
                        fprintf(f, "\t\t\t:val_%s;\n\n", (Ibase[i]->
itemlist)[k]->itemname);
                        }
                        else
```

```
                                fprintf(f, "\t\t\t:val_%s,\n",
(Ibase[i]-> itemlist)[k]->itemname);
                        ++k;
                        ++count;
                }
                ++i;
        }

        i = 0;
        while (Iagglist[i] != NULL)
        {
                if (count == Var_Num - 1)
                {
                        fprintf(f, "\t\t\t:val_%s;\n\n", Iagglist[i]->
itemname);
                }
                else
                        fprintf(f, "\t\t\t:val_%s,\n", Iagglist[i]->
itemname);

                ++i;
                ++count;
        }

      i = 0;
        while (Ibase[i] != 0)
        {
                k = 0;
                while ((Ibase[i]->itemlist)[k] != NULL)
                {
                        if (!strcmp((Ibase[i]->itemlist)[k]->iformat,
"CHARACTER"))
                        {
                        fprintf(f, "\tval_%s.arr[val_%s.len] =
\'\\0\';\n", (Ibase[i]->itemlist)[k]->itemname, (Ibase[i]->itemlist)[k]-
>itemname);
                        }
                        ++k;
                }
                ++i;
        }
      fprintf(f, "\n");

}

/***********************************************************************
cursor_declaration(f)

Description:      Delare the cursor for fetching data from factbase.
***********************************************************************/

cursor_declaration(f)
FILE *f;
{
      int i,k, count=0;

      fprintf(f, "\tEXEC SQL WHENEVER SQLERROR GOTO sqlerror;\n");

      fprintf(f, "\tEXEC SQL DECLARE factbase CURSOR FOR\n");

      fprintf(f, "\n\t\tSELECT DISTINCT\n");

      i = 0;
      while (Ibase[i] != NULL)
      {
            k = 0;
            while ((Ibase[i]->itemlist)[k] != NULL)
            {
                  if (count == Var_Num -1)
                  {
                  fprintf(f, "\t\t\t%s\n", (Ibase[i]->itemlist)[k]->
itemname);
                  }
                  else
                        fprintf(f, "\t\t\t%s,\n", (Ibase[i]-
>itemlist)[k]-> itemname);
                  ++k;
                  ++count;
            }
            ++i;
      }

      i = 0;
      while (Iagglist[i] != NULL)
      {
            if (count == Var_Num - 1)
            {
                  fprintf(f, "\t\t\t%s\n", Iagglist[i]->itemname);
            }
            else
                  fprintf(f, "\t\t\t%s,\n", Iagglist[i]->itemname);

            ++i;
            ++count;
      }

      fprintf(f, "\t\tFROM %s_factbase;\n\n", rule_id);

}


/***********************************************************************
host_variable_declaration(f)

Description:      Generate the Pro*C host variables delaration section.
***********************************************************************/

host_variable_declaration(f)
FILE *f;
```

```
{
      int i;
      int k;

      fprintf(f, "\n\tEXEC SQL BEGIN DECLARE SECTION;\n");

      i=0;
      while(Ibase[i] != NULL)
      {
            k = 0;
            while((Ibase[i]->itemlist)[k] != NULL)
            {
                  if (!strcmp((Ibase[i]->itemlist)[k]->iformat,
"INTEGER"))
                  {
                        fprintf(f,"\t\tint      val_%s;\n", (Ibase[i]->
itemlist)[k]->itemname);
                  }
                  else if (!strcmp((Ibase[i]->itemlist)[k]->iformat,
"CHARACTER"))
                  {
                        fprintf(f, "\t\tVARCHAR  val_%s[%d];\n",
(Ibase[i]->itemlist)[k]->itemname, (Ibase[i]->itemlist)[k]->ilength +
1);
                  }
                  else if (!strcmp((Ibase[i]->itemlist)[k]->iformat,
"REAL"))
                  {
                        fprintf(f,"\t\tfloat   val_%s;\n", (Ibase[i]-
>itemlist)[k]->itemname);
                  }
                  else
                        fprintf(f, "\t\t Pro*C undefined type\n");

                  ++Var_Num;

                  ++k;
            }

            ++i;
      }



      k = 0;
      while (Iagglist[k] != NULL)
      {
            if (!strcmp(Iagglist[k]->iformat, "INTEGER"))
            {
                   fprintf(f,"\t\tint      val_%s;\n", Iagglist[k]-
>itemname);
            }
            else if (!strcmp(Iagglist[k]->iformat, "CHARACTER"))
            {
                   fprintf(f, "\t\tVARCHAR  val_%s[%d];\n",
Iagglist[k]->itemname, Iagglist[k]->ilength + 1);
                        }
                        else if (!strcmp(Iagglist[k]->iformat, "REAL"))
                        {
                                fprintf(f,"\t\tfloat   val_%s;\n",
Iagglist[k]->itemname);
                        }
                        else
                                fprintf(f, "\t\t Pro*C undefined
type\n");

                  ++Var_Num;

                        ++k;
       }


      fprintf(f, "\tEXEC SQL END DECLARE SECTION;\n\n");

}


/***********************************************************************
print_infix(f, node, first)

FILE *f
            The *.pc file for storing the Pro*C code of the rule;
tnode *node
            The node of the rule-tree;
int *first
            For operation precedence;
Description:
            Traverse the rule-tree and print the needed node information
            for the Pro*C rule condition and actions.
***********************************************************************/

print_infix(f, node, first)
FILE *f;
tnode *node;
int *first;
{
    tIagg **list;
    tItab **list2;
    int i, flag;


    if (node->ntype == N_IF)
       {
      fprintf(f, "\tif (");

      i = 0;
      while ((node->children)[i] != NULL)
      {
            print_infix(f, (node->children)[i], first);
            ++i;
      }

      fprintf(f, ")\n\t{ ");
       return;
       }

    if(node->ntype == N_UPDATE_DIR)
        {
        return;
        }

    if(node->ntype == N_FUNCT &&
        (!strcmp((char*)(node->nvalue), "exist") ||
         !strcmp((char*)(node->nvalue), "count") ||
         !strcmp((char*)(node->nvalue), "sum") ||
         !strcmp((char*)(node->nvalue), "max") ||
         !strcmp((char*)(node->nvalue), "min") ||
         !strcmp((char*)(node->nvalue), "avg")))
        {
        for(list = Iagg; *list != NULL; list++)
            if(!strcmp(node->func_id, (*list)->f_dk))  /* agg-funct
identifer */
                break;

        fprintf(f, " val_%s", node->func_id);

        return;
        }

    if(node->ntype == N_FUNCT)
        {
        fprintf(f, "\t\t%s(", (char *)(node->nvalue));
        for(i=0; node->children[i] != NULL; i++)
            {
            if(i != 0)
                fprintf(f, ",");
            print_infix(f, node->children[i],first);
            }
        fprintf(f, ");\n");
        return;
        }
    if(node->valuetype == NOT)
        {
        fprintf(f,"!");
        print_infix(f, node->children[0], first);
        return;
        }

    if(node->ntype == N_OPER)  /* for operation like AND, OR, *, + ,-,
etc */
        {

      if ((!strcmp((char *)node->nvalue, ":="))&&
                 ((node->children)[0]->ntype == N_ITEM)&&
                 ((node->children)[1]->valuetype == STRING))
```

```
             {
                fprintf(f, "\n\t\tstrcpy(val_%s.arr, \"%s\");\n", (node-
>children)[0]->nvalue, (node->children)[1]->nvalue);

                fprintf(f, "\t\tval_%s.len = strlen(\"%s\");\n\n",
(node->children)[0]->nvalue, (node->children)[1]->nvalue);

            return;
             }

        if(*first != 1)
            fprintf(f, "(");
        flag = *first;
        *first = 0;
        print_infix(f, node->children[0], first);

      if (!strcmp((char *)node->nvalue, "AND"))
            fprintf(f, " && ");
      else if (!strcmp((char *)node->nvalue, "OR"))
            fprintf(f, " || ");
      else
            fprintf(f, " %s ", (char *)(node->nvalue));

        print_infix(f, node->children[1], first);
        if(flag != 1)
            fprintf(f, ")");
        return;
        }
      if(node->ntype == N_ITEM)
        {
        for(list2 = Itab; *list2 != NULL; list2++)
            if(!strcmp((*list2)->applname, node->applname) &&
               !strcmp((*list2)->oeprname, node->oepr) &&
               !strcmp((*list2)->itemname, (char *)(node->nvalue)))
                break;

        fprintf(f, "val_%s ", (*list2)->itemname);

        return;
        }
    if(node->valuetype == INTEGER)
        fprintf(f, " %d ", *(int *)(node->nvalue));
    if(node->valuetype == REAL)
        fprintf(f, " %f ", *(double *)(node->nvalue));
    if(node->valuetype == STRING)
        fprintf(f, " \"%s\" ", (char *)(node->nvalue));
}


/************************************************************************
include_user_defined_routine(f, node, flag)

FILE *f
            The *.pc file for storing Pro*C code of the rule;
tnode *node
```

```
            The node of the rule-tree;
int *flag
            For avoiding repeated including files;
Description:
            Include the files storing user-defined routines.
***********************************************************************/

include_user_defined_routine(f, node, flag)
FILE *f;
tnode *node;
int *flag;
{
    char *ret_type, *coded, *loc;
    int i;

    for(i=0; node->children[i] != NULL; i++)
        include_user_defined_routine(f, node->children[i], flag);
    if(node->ntype == N_FUNCT &&
        (strcmp((char*)(node->nvalue), "exist") &&
         strcmp((char*)(node->nvalue), "count") &&
         strcmp((char*)(node->nvalue), "sum") &&
         strcmp((char*)(node->nvalue), "max") &&
         strcmp((char*)(node->nvalue), "min") &&
         strcmp((char*)(node->nvalue), "avg")))
        {

      /* get the user-define routines info from MDB */
        get_detailed_funct(node->applname, (char *)(node->nvalue),
&coded, &loc,&ret_type);

      fprintf(f, "\n#include \"%s\"\n\n", loc);

        free(loc);
        free(coded);
        free(ret_type);
        }
}

/***********************************************************************
print_update_query(f)

FILE *f
            The *.pc file for storing Pro*C cod eof the rule;
Description:
            Generate the Pro*C SQL update statement for the rule.
***********************************************************************/

print_update_query(f)
FILE *f;
{
    tIset **list;
    tIoepr  **list2; /* store info for updated OEPR */
    int i;

    for(list = Iset; *list != NULL; list++)  /* for each ind. Query */
```

```
        {
        for(i = 0; (*list)->appllist[i] != NULL; i++)
            {
            if(!strcmp((*list)->appllist[i], "ORDER_PROCESSING"))
                {
                for(list2 = Ioepr; *list2 != NULL; list2++)
                    if(!strcmp((*list2)->applname, (*list)->appllist[i])
&&
                       !strcmp((*list2)->oeprname, (*list)->objlist[i]))
                        break;


                if(*((*list)->A_of_obj[i]) != NULL)
                    {
                      if ((*list)->Ind_of_Temp[i] == 1)
                          Create_Update_Template(*list2, (*list)->
A_of_obj[i], f);
                      else
                          ERROR("the update is not supported\n");

                    }
                }
      }
    }
}


/************************************************************************
Create_Update_Template(oepr, Ajo, f)

tIoepr *oepr
            The structure for storing OE/PR information;
char **Ajo
            The data items being updated;
FILE *f
            The *.pc file for storing Pro*C code of the rule;
Description:
            Generate the SQL update for the non-key item updation.
************************************************************************/

Create_Update_Template(oepr, Ajo, f)
tIoepr *oepr;
char **Ajo;
FILE *f;
{
    tItab **list2;
    int i;
    char **list, **list3;

    list = Ajo;
    fprintf(f, "\n\t\tEXEC SQL UPDATE %s \n",  oepr->lc_oeprname);
    fprintf(f, "\t\t\tSET %s = ", *list);

    for(list2 = Itab; *list2 != NULL; list2++)
        if(!strcmp(oepr->oeprname, (*list2)->oeprname) &&
           !strcmp(oepr->applname, (*list2)->applname) &&
           !strcmp(*list, (*list2)->itemname))
            break;
    fprintf(f, ":val_%s\n", (*list2)->itemname);

    fprintf(f, "\t\t\tWHERE %s = :val_%s", oepr->key_of_oepr[0], oepr->
key_of_oepr[0]);

    for(i=1; oepr->key_of_oepr[i] != NULL; i++)
        {
        fprintf(f, "\n\t\t\tAND %s = :val_%s", oepr->key_of_oepr[i],
oepr->key_of_oepr[i]);
        }

    fprintf(f, ";\n");
}
```

```
/***********************************************************************

            Filename: timer.c

            Description: Parse the trigger information and trigger the
                       local-rules according to their time-trigger
                       specification.

***********************************************************************/

#include <time.h>
#include <stdio.h>
#include <string.h>
#include "lexshell.h"
#include "lexshell.c"

#define  ERROR(str)  {printf("\n\n%s\n",str);exit(1);}

typedef struct ttrig
{
    char *ruleid;
    char *time;
    int  delta[5];  /* for non-absolute time trigger */
    int  triggered; /* there may be many points it be triggered,but only
once */
    int  absolute; /* when_time_is or every_time */
    int  year;
    int  month;
    int  day;
    int  hour;
    int  minute;
}ttrig;


struct tm    *cur_time;

tptr    *new_list ()

{
    tptr    *new;

    new = (tptr *)malloc(sizeof(tptr));
    new[0] = NULL;

    return (new);
}


tptr *add_elem (count,list,item)
int    count;
tptr   *list,item;

{
    int    j;
    list = (tptr *) realloc (list,sizeof(tptr)*(count+2));
    list[count]=item;
    list[count+1]=NULL;

    return (list);

}


ttrig **trigs = NULL;


/*******************************************************************
main()

Description:    The main function for timer of the local-rule.
            Parse the trigger information in trig_mes.doc and every
            10 seconds to check which rules should be triggered.
*******************************************************************/

main()
{
      FILE *trigfile;
        time_t    time_offset;

      trigfile = fopen("trig_mes.doc", "r");

      time_offset = time(NULL);

      cur_time = localtime(&time_offset);  /* # of ms to time struct */

      parse_time_trigger(trigfile);

      fclose(trigfile);

      while (1==1)
      {
            trigger_rules();

            sleep(10);

            printf("\n...... Checking who should be triggered
......\n\n");
      }

}

/*******************************************************************
trigger_rules()

Description:
            Check each rule to see whether its event time equals to the
            current time, if so, trigger the rule. For un-absolute time-
            triggered rule, we need to update its next event-time.
*******************************************************************/
```

```
trigger_rules()
{
    time_t    time_offset;
    int i,c;
    char    rule_id[256],command[256];

    time_offset = time(NULL);

    cur_time = localtime(&time_offset);  /* # of ms to time struct */

    printf("\n%d/%d/%d %d:%d:%d\n",cur_time->tm_mon,cur_time->tm_mday,
cur_time->tm_year+1900,cur_time->tm_hour,cur_time->tm_min,cur_time->
tm_sec);

    i = 0;
    while (trigs[i] != NULL)
    {

        if ((trigs[i]->hour == cur_time->tm_hour)&&
            (trigs[i]->minute == cur_time->tm_min))
        {
            if (!trigs[i]->triggered)
            {

              printf("\n\n The **** %s **** is triggered!\n\n",
trigs[i]->ruleid);
              printf("Nothing happens if the condition is false,
otherwise ...\n\n");

              sprintf(command, "../opsrule/%s", trigs[i]->ruleid);
              system(command);

              trigs[i]->triggered = TRUE;

              if (!trigs[i]->absolute)
                  adjust_next_event_time(&(trigs[i]));

            }
              else if ((cur_time->tm_sec >= 50)&&(cur_time->tm_sec <=
59))
                  trigs[i]->triggered = FALSE;

          }
        else if (((trigs[i]->hour != cur_time->tm_hour)||
                    (trigs[i]->minute != cur_time->tm_min))
                  &&(trigs[i]->triggered))

                trigs[i]->triggered = FALSE;

        ++i;

    }

}
```

```
/***********************************************************************
adjust_next_event_time(trig)

ttrig **trig
            The ttrig structure for a specific rule;
Description:
            Adjust the next event time of this rule.
***********************************************************************/

adjust_next_event_time(trig)
ttrig **trig;
{

      if (((*trig)->hour + (*trig)->delta[3]) >= 24)
             (*trig)->hour = (*trig)->hour + (*trig)->delta[3] - 24;
        else
             (*trig)->hour = (*trig)->hour + (*trig)->delta[3];


        if (((*trig)->minute + (*trig)->delta[4]) >= 60)
             (*trig)->minute = (*trig)->minute + (*trig)->delta[4] - 60;
        else
             (*trig)->minute = (*trig)->minute + (*trig)->delta[4];

}

/***********************************************************************
init_next_event_time(trig)

ttrig **trig
            The ttrig structure for the rule;
Description:
            Initialize the next event time for the rule.
***********************************************************************/

init_next_event_time(trig)
ttrig **trig;
{


       if ((cur_time->tm_hour + (*trig)->delta[3]) >= 24)
             (*trig)->hour = cur_time->tm_hour + (*trig)->delta[3] - 24;
        else
             (*trig)->hour = cur_time->tm_hour + (*trig)->delta[3];


        if ((cur_time->tm_min + (*trig)->delta[4]) >= 60)
             (*trig)->minute = cur_time->tm_min + (*trig)->delta[4] -
60;
        else
             (*trig)->minute = cur_time->tm_min + (*trig)->delta[4];

}
```

```
/*********************************************************************
parse_time_trigger(in)

FILE *in
            The trig_mes.doc file which store the trigger info for
rules;

Description:
            Parsing the trig_mes.doc file to store all trigger
informations
            in the ttrig structures.
*********************************************************************/

parse_time_trigger(in)
FILE *in;
{
    ttrig *trig;
    int   token;
    tptr  value;
    char  prev_char = '\0';
    int count;

    trigs = (ttrig **)new_list();

    token = lexical(in,&value,&prev_char);

    count = 0;
    while (token == WORDSYM)
        {
          trig = (ttrig *)malloc(sizeof(ttrig));
          trig->ruleid = (char *)value;

          token = lexical(in,&value,&prev_char); /* is */
          token = lexical(in,&value,&prev_char); /* triggered */

          if (!cmpstr((char *)value,"triggered"))
          {
                trig_info(in,&token,&value,&prev_char,trig);

                trigs = (ttrig **)add_elem(count, (tptr *)trigs,
(tptr)trig);

                ++count;
          }
          else
                ERROR("trig_msg.doc errror\n");
      }


    return (TRUE);
}

/*********************************************************************
trig_info(in,token,value,prev_char,trig)
```

```
FILE *in
            The trig_mes.doc file;
int *token
            The token output from lexical analyzer;
tptr *value
            The value output from lexical analyzer;
char *prev_char
            The previous token;
ttrig *trig
            The ttrig structure for storing trigger information;
Description:
            Get the detailed trigger information from trig_mes.doc.
***********************************************************************/

trig_info(in,token,value,prev_char,trig)
FILE *in;
int *token;
tptr *value;
char *prev_char;
ttrig *trig;
{
    FILE *out;
    int i,j,absolute,val[5],count;
    char *fname,command[256];

    *token = lexical(in,value,prev_char);  /* by */
    *token = lexical(in,value,prev_char);  /* time */
    *token = lexical(in,value,prev_char);  /* using */

    *token = lexical(in,value,prev_char);
    trig->time = (char *)malloc(strlen(*value));
    strcpy(trig->time, (char *)(*value));
    trig->absolute = TRUE;
    trig->year = -1;
    trig->month = -1;
    trig->day = -1;
    trig->hour = 0;
    trig->minute = 0;


    for (i=0;i<5;i++)
         val[i] = 0;

    i=0;
    if (trig->time[0] == '+')
    {
        i++;
        trig->absolute = FALSE;
    }

    printf("%s\n", trig->time);

    for (j=0;trig->time[i] != '\0';i++)
    {
         if (trig->time[i] == '?')
         {
             val[j] = -1;
         }
         else if ((trig->time[i] >= '0') && (trig->time[i] <= '9'))
             val[j] = val[j] * 10 + trig->time[i] - '0' ;
         else
             j++;
    }

    if (trig->absolute)
    {
        trig->year = val[2];
        trig->month = val[0];
        trig->day = val[1];
        trig->hour = val[3];
        trig->minute = val[4];
    }
    else  /* for the rule which is triggered by non-absolute time */
    {
      trig->delta[0] = val[2];
      trig->delta[1] = val[0];
      trig->delta[2] = val[1];
      trig->delta[3] = val[3];
      trig->delta[4] = val[4];

      init_next_event_time(&trig);

    }

    *token = lexical(in,value,prev_char);

}
```

```
/***********************************************************************

            Filename:  arrange.c

            Description: Rearrange the condition and actions of the
                         global-rule tree to obtain a set of condition
                         and actions processed in a single system.

***********************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include "message.h"

#define MAX_NODE 50
#define  ERROR(str)  {printf("\n\n%s\n",str);exit(1);}
extern  tnode* rule_tree;

int node_matrix[MAX_NODE][MAX_NODE]; /* used fo rrepresenting the
precedence relationship betwee the nodes which are direct sons of the
root of the ruel tree, this is the result of rule serialization; and
later partition is based on this matrix. */


/***********************************************************************
tptr *insert_elem(int pos, tptr *list, tptr item)
int pos
                The position before which the item being inserted.
tptr *list
                The list the item be inserted.
tptr item
                The item itself.

Description:    Insert the item into the list at postion pos.

***********************************************************************/

tptr *insert_elem(int pos, tptr *list, tptr item)
{
     int count = 0;
     int i;

     while (list[count] != NULL)
          ++count;

     list = (tptr *) realloc(list, sizeof(tptr)*(count+2));

     i = count;

     while (i > pos)
     {
         list[i] = list[i-1];
         --i;
     }

     list[pos] = item;
     list[count+1] = NULL;

     return list;
}

/***********************************************************************
delete_elem (int pos, tnode *root)

int pos
                Which children of this tree node.
tnode *root
                The root node of the rule_tree.

Delete the child node of the root node at position pos.
***********************************************************************/

delete_elem (int pos, tnode *root)
{

   int i;

   i  = pos;

   while ((root->children)[i] != NULL)
   {
      (root->children)[i] = (root->children)[i+1];
        ++i;
   }

}

/***********************************************************************
int card(tptr *list)

tptr *list
            The list being calculated.

Calculate the total number of elements in the list.
***********************************************************************/

int card(tptr *list)
{
     int i = 0;

     while (list[i] != NULL)
          ++i;

     return(i);
}
```

```
/***********************************************************************
getout_rewrite (root, O_node, O_pos, Ai_node, i, o_node, Ti_node)

tnode *root
            The root node of the tree.
tnode *O_node
            The node which contains sub-nodes which will be moved out.
int O_pos;
            The position of the above node in the children list of the
root.
tnode *Ai_node
            The sub-node contained in O_node and it will be removed out.
int i;
            The position of Ai_node in the children list of O_node.
tnode **o_node;
            The new node which inserted under the root for the removed
node
            Ai_node.
tnode **Ti_node;
            The node for storing the temporary variable for storing Ai.

Rewrite the rule tree root(..., O(v, e(..,Ai,...)),...) as root(..., o,
O(v, e(..., Ti, ...)), ...).

***********************************************************************/

getout_rewrite (root, O_node, O_pos, Ai_node, i, o_node, Ti_node)
tnode *root;
tnode *O_node;
int O_pos;
tnode *Ai_node;
int i;
tnode **o_node;
tnode **Ti_node;
{
      static int tempcount = 0;
      char *tempname;

      /* create name for temp variable */

        ++tempcount;
        tempname = (char *)malloc(9);
        sprintf(tempname, "TEMP_%d", tempcount);

      /* create nodes for Ti := Ai */

        *Ti_node = create_tnode(N_TEMP,IDENT,tempname,NILS, NILS,
NILS,0);

      *o_node = create_tnode(N_OPER, ASSIGN, ":=", NILS, NILS, NILS,2);

        (*o_node)->children = (tnode **)add_elem(0, (tptr *)(*o_node)->
children, (tptr)*Ti_node);
```

```
      (*o_node)->children = (tnode **)add_elem(1, (tptr *)(*o_node)->
children, (tptr)Ai_node);

      /* insert this new nodes into root children list */

      root->children = (tnode **)insert_elem (O_pos, (tptr *)root->
children, (tptr)*o_node);

      ++(root->nb_children);

      /* substitute Ai with Ti */

      (O_node->children)[i] = *Ti_node;

}

/***********************************************************************

put_in_rewrite (root, o_node, o_pos, O_node, i)

tnode *root
            The root node of the tree;
tnode *o_node
            The node which will be put back into O_node;
int o_pos
            The position of o_node in the children list of root;
tnode *O_node
            The node under which the o_node will be put in;
int i
            The position in the children list of O_node where o_node
will
            be put in.

Rewrite root(...,o, O(v, e(...,Ti,...)),...) as root(...,O(v,
e(...,Ai,...)),...).

***********************************************************************/

put_in_rewrite (root, o_node, o_pos, O_node, i)
tnode *root;
tnode *o_node;
int o_pos;
tnode *O_node;
int i;
{

      (O_node->children)[i] = (o_node->children)[1];

        delete_elem (o_pos, root);

        --(root->nb_children);
}


/***********************************************************************
```

```
int belongto (tptr elem, tptr *list)

tptr elem
            The element to be checked;
tptr *list
            The list to be checked.

Check whether the element is in the list or not, return TRUE or FALSE.

***********************************************************************/

int belongto (tptr elem, tptr *list)
{

      int i = 0;

      while ((list[i] != NULL)&&(strcmp(elem, list[i])))
            ++i;

      if (list[i] == NULL) return FALSE;
      else return TRUE;
}

/***********************************************************************
serialize_assign(root, O_node, O_pos)

tnode *root
            The root node of the tree;
tnode *O_node
            The node which will be serialized;
int *O_pos
            The position of O_node in the children list of the root
node;

Serialize the direct child node of the root node to guranttee each node
under the root node will be executed in the same application system.
***********************************************************************/

serialize_assign(root, O_node, O_pos)
tnode *root;
tnode *O_node;
int *O_pos;
{
        int n,i,o_pos;
        tnode *o_node, *Ti_node, *Ai_node;

      n = (O_node->children)[1]->nb_children;

      if (n == 0) return;

      for (i = 0; i < n; ++i)
      {
            Ai_node = ((O_node->children)[1]->children)[i];

            if (
                (card(Ai_node->sysnames) > 1)
                ||((Ai_node->executed_in != NULL)
                    &&(O_node->executed_in !=NULL)
                    &&strcmp(Ai_node->executed_in, O_node->executed_in))
                ||((Ai_node->executed_in != NULL)
                  &&(O_node->executed_in ==NULL))
                ||(((Ai_node->sysnames)[0] != NULL)&&(O_node->
executed_in !=NULL)&&strcmp((Ai_node->sysnames)[0], O_node->
executed_in))
                ||(((Ai_node->sysnames)[0] != NULL)&&(O_node->
executed_in ==NULL))
               )
            {

                  getout_rewrite (root, (O_node->children)[1], *O_pos,
Ai_node, i, &o_node,&Ti_node);

                  if ((Ai_node->executed_in == NULL)&&(belongto(O_node-
>executed_in, Ai_node->sysnames)))
                        o_node->executed_in = O_node->executed_in;
                  else
                        o_node->executed_in = NULL;

                  o_node->sysnames = Ai_node->sysnames;

                  Ti_node->executed_in = NULL;

                  o_pos = *O_pos;

                  serialize_assign (root, o_node,&o_pos);

                  *O_pos = o_pos + 1;

                  if ((O_node->executed_in == NULL)
                     ||((o_node->executed_in != NULL)&&
                     (!strcmp(o_node->executed_in, O_node->
executed_in)))
                  /*    ||(Ai_node->executed_in == NULL) */
                      ||((Ai_node->executed_in != NULL)&&
                        !strcmp(Ai_node->executed_in, O_node->
executed_in)))
                  {
                        put_in_rewrite(root, o_node, o_pos, (O_node->
children)[1], i);
                        *O_pos = o_pos;

                        if (((O_node->children)[1]->children)[i]->
executed_in != NULL)
                            O_node->executed_in = ((O_node->
children)[1]->children)[i]->executed_in;
                  }
            }
      }

      if (O_node->executed_in == NULL)
      {
            O_node->sysnames = (char **)new_list();
      }
      else
      {
            O_node->sysnames = (char **)new_list();
            O_node->sysnames = (char **)add_elem(0, (tptr *)O_node-
>sysnames, O_node->executed_in);
      }

}

/*************************************************************************
union_list (list1, list2)

char*** list1
            The list where the final union result will be stored;
char*** list2
            One of the list being unioned;

Union the two sets list1 and list2 and the result is stored in list1.
*************************************************************************/

union_list (list1, list2)
char*** list1;
char*** list2;
{
    int i = 0;
    int k,j;

    k = 0;
    while ((*list1)[k] != NULL)
         ++k;

    i = 0;
    while ((*list2)[i] != NULL)
          {
           j = 0;
           while (((*list1)[j] != NULL)&&(strcmp((*list1)[j],
(*list2)[i])))
                 ++j;


           if ((*list1)[j] == NULL)
              {
              *list1 = (char **)add_elem(j, (tptr *)*list1,
(tptr)(*list2)[i]);
              }

           ++i;
          }

}

/*************************************************************************
```

```
preorder_determine_SysO (O_node)

tnode *O_node
            The node which its sysnames will be determined;

Determine the all systems in which the node (a sub-tree, includng all
its child nodes) to be executed.

*********************************************************************/

preorder_determine_SysO (O_node)
tnode *O_node;
{
        int i,k;


        i = 0;
      while ((O_node->children)[i] != NULL)
      {
               preorder_determine_SysO((O_node->children)[i]);


             union_list (&(O_node->sysnames), &((O_node->children)[i]-
>sysnames));
               ++i;
        }


      if (O_node->executed_in != NULL)
            {
            k = 0;
            while (((O_node->sysnames)[k] != NULL)&&strcmp((O_node-
>sysnames)[k], O_node->executed_in))
            {
            ++k;
            }

            if ((O_node->sysnames)[k] == NULL)
                        O_node->sysnames = (char **)add_elem(k, (tptr
*)O_node->sysnames,(tptr)O_node->executed_in);
            }

}

/*********************************************************************
int exists_smallest_Oj(i,j,count,list)

int i
            The current node position;
int *j
            The position to be found out which is nearest to i;
int count
            The total number of nodes being checked;
tnode **list
            The children node list of the root node;
```

```
Find out the nearest node j such that App(Oj) belongto Sys(Oi), return
TRUE or FALSE.
************************************************************************/

int exists_smallest_Oj(i,j,count,list)
int i;
int *j;
int count;
tnode **list;
{
      int k;


      *j = i + 1;

      if (*j == count)
          return (FALSE);

      while (list[*j] != NULL)
      {
            if (list[*j]->executed_in != NULL)
            {
                  k = 0;
                  while(((list[i]->sysnames)[k] != NULL)&&
(strcmp(list[*j]->executed_in, (list[i]->sysnames)[k])))
                        ++k;

                  if ((list[i]->sysnames)[k] != NULL)
                  {
                        return (TRUE);
                  }
            }
            *j = *j + 1;
      }

      return (FALSE);

}

/************************************************************************
reverse_order_adjust_AppO (list)

tnode **list
            The children node list of the root;

Adjust the executed_in(the system where the node is executed) in a
reverse order.
************************************************************************/

reverse_order_adjust_AppO (list)
tnode **list;
{
      int i = 0, j;
      int count = 0;
      while (list[count] != NULL)
            ++count;

      i = count - 1;

      while (i >= 0)
      {
            if ((list[i]->executed_in == NULL)&&(card(list[i]->sysnames)
== 1))
                  list[i]->executed_in = (list[i]->sysnames)[0];

            /* the case when sys > 1 */

            if ((list[i]->executed_in == NULL)&&(card(list[i]->sysnames)
> 1)&&(exists_smallest_Oj(i, &j, count, list)))
                  list[i]->executed_in = list[j]->executed_in;

            --i;
      }

}

/***********************************************************************
arrange_rulelist(tnode* root)

The main function for the rule-tree rearrangement and partition.

***********************************************************************/

arrange_rulelist(tnode* root)
{
      static int i = 0;
        int node_num;

      printf("The %d RULE:\n\n", i+1);
      arrange_rule(root, &node_num);
      partition_rule(root, node_num);

      rule_tree = root;
}


/***********************************************************************
arrange_rule(tnode *root, int *node_num)

tnode *root
            The root node of the tree;
int *node_num
            The index for the node.

The major function for rule tree rearrangement.
***********************************************************************/

arrange_rule(tnode *root, int *node_num)
```

```
{
      int i = 0,j;
      int pos = 0;
/*
      root = create_test_tree();
*/

      printf("\n\nTHE ORIGINAL RULE:\n");
      display_rule_tree(root,0);

      preorder_determine_SysO (root);

      printf("\n\nPREORDER DETERMINE SYSO:\n");
      display_rule_tree(root,0);

      reverse_order_adjust_AppO (root->children);

      printf("\n\nREVERSE ADJUST APPLO TREE!!\n");
      display_rule_tree(root,0);

      while ((root->children)[pos] != NULL)
      {
            serialize(root, (root->children)[pos], &pos);
            ++pos;
      }

      printf("\n\nLAST REARRANGED TREE!!\n");
      display_rule_tree(root,0);

      *node_num = pos;

      generate_matrix_of_precedence(root->children, pos);

      printf("\n\nTHE MATRIX OF PRECEDENCE!!\n");
      for (i=0; i<pos; i++)
      {
          for (j=0; j<pos; j++)
          {
               printf("%d ", node_matrix[i][j]);
          }
          printf("\n");
        }
}


/************************************************************************
int found_user(char *elem, tnode **list)

char *elem
            The element(eg:item, run_time item, temp variable) being
checked
tnode **list
            The node list under the node being checked.
```

```
Check the node (sub-tree) to see if the element is used among this sub-
tree.

***********************************************************************/

int found_user(char *elem, tnode **list)
{

      int i, found;

      i = 0;
        found = FALSE;

      while ((!found)&&(list[i] != NULL))
      {
            if ((list[i]->ntype == N_ITEM)||(list[i]->ntype == N_TEMP))
            {
                  if (!strcmp(elem, list[i]->nvalue))
                      found = TRUE;
                  else
                      ++i;
            }
            else
            {
                  if (list[i]->children != NULL)
                      found = found_user(elem, list[i]->children);
                  else
                      found = FALSE;

                  ++i;
              }
      }

      return found;

}

/***********************************************************************
determine_precedence_for_temp_variable(lvalue_node, list, i, n)

ltnode *lvalue_node
            The node which is the left-child of the assignment node;
tnode **list
            The node list which are the direct children of the root;
int i
            The position of this assignment node in the root children
list;
int n
            The total number of children of the root;

Establish precedence relationship between the assignment node which
assign toa temporary variable with other nodes which maybe use this
temporary variable.
***********************************************************************/
```

```
determine_precedence_for_temp_variable(lvalue_node, list, i, n)
tnode *lvalue_node;
tnode **list;
int i;
int n;
{
      int k, found;

      k = i+1;

      found = FALSE;

      while ((!found)&&(k < n))
        {
              if (list[k]->children == NULL)
              {
                   found = FALSE;
                   ++k;
              }
              else
              {
                  if (found_user(lvalue_node->nvalue, list[k]-
>children))
                       found = TRUE;
                  else
                  {
                       found = FALSE;
                       ++k;
                  }
             }
        }

        if (found == TRUE)
        {
              node_matrix[i][k] = -1;
              node_matrix[k][i] = 1;
        }
        else
              ERROR("error for temp usage\n");
}


/***********************************************************************
backward_determine_precedence_for_item(lvalue_node,i, ifpos, list)

tnode *lvalue_node
            The node which is the left-child of the assignment node;
tnode **list
                The node list which are the direct children of the root;
int i
                The position of this assignment node in the root
                children list;
int ifpos
                The position of the IF node in the root children list;
```

```
Establish precedence relationship between the assignment node which
assign to a data item (update the data item) with other nodes which has
already used this data item before the update.

*********************************************************************/

backward_determine_precedence_for_item(lvalue_node,i, ifpos, list)
tnode *lvalue_node;
int i;
int ifpos;
tnode **list;
{
      int k, found;

      k = i - 1;
      found = FALSE;

      while ((!found)&&(k > ifpos))
      {
            if ((list[k]->valuetype == ASSIGN)&&
               !strcmp((list[k]->children)[0]->nvalue, lvalue_node-
>nvalue))
            {
                  found = TRUE;  /* we stop when meet with the same
assign */
            }
            else if ((list[k]->children != NULL)
                  &&(found_user(lvalue_node->nvalue, list[k]-
>children)))
            {
                  node_matrix[k][i] = -1;
                  node_matrix[i][k] = 1;

                  --k;
            }
            else
            {
                  found = FALSE;  /* other case, continue checking */
                  --k;
            }
      }

}


/*********************************************************************

forward_determine_precedence_for_item(lvalue_node,i, ifpos, list)

tnode *lvalue_node
                The node which is the left-child of the assignment node;
tnode **list
                The node list which are the direct children of the root;
int i
```

```
                The position of this assignment node in the root
                children list;
int n
                The length the root children list;

Establish precedence relationship between the assignment node which
assign to a data item (update the data item) with other nodes which will
use this data item after the update.

************************************************************************/

forward_determine_precedence_for_item(lvalue_node, i, n, list)
tnode *lvalue_node;
int i;
int n;
tnode **list;
{
      int k, found, exist_user;

      k = i + 1;
      found = FALSE;
      exist_user = FALSE;

        while ((!found)&&(k < n))
        {

              /* case 1: find the same assign */

              if ((list[k]->valuetype == ASSIGN)&&
                  !strcmp((list[k]->children)[0]->nvalue, lvalue_node-
>nvalue))
              {

                  if (exist_user == FALSE)  /* when no exist user
between two same assignment */
                  {
                        node_matrix[i][k] = -1;
                        node_matrix[k][i] = 1;
                  }

                  found = TRUE;  /* we stop checking when we meet the
same assignment */

              }

              /* case 2: find the update-directive */

              else if ((list[k]->ntype ==
N_UPDATE_DIR)&&!strcmp(list[k]->executed_in, lvalue_node->applname))
              {
                     node_matrix[i][k] = -1;
                     node_matrix[k][i] = 1;

                     found = TRUE;
                     /* we do not need to continue check */
              }

              /* case 3: find the user of the variable been assigned */

              else if ((list[k]->children != NULL)
                       &&!((list[k]->valuetype == ASSIGN)
                       &&(!strcmp((list[k]->children)[0]->nvalue,
lvalue_node->nvalue)))
                       &&(found_user(lvalue_node->nvalue, list[k]->
children)))
              {
                       node_matrix[i][k] = -1;
                       node_matrix[k][i] = 1;
                       ++k;

                       exist_user = TRUE;

                       /* here we need to continue check */
              }

              /* other cases */

              else
              {
                        found = FALSE;
                        ++k;
              }
      }

}

/************************************************************************

determine_precedence_for_ifnode_and_action(ifpos, n)

int ifpos
            The position of the IF node;
int n
            The length of the root children list;

Establish the precedence relationship between the IF node and the action
nodes in the rule tree.

************************************************************************/

determine_precedence_for_ifnode_and_action(ifpos, n)
int ifpos;
int n;
{
        int i,k,found;

        for (k = ifpos+1; k < n; ++k)
        {
                found = FALSE;
                i = ifpos;
                while ((!found)&&(i < k))
                {
                        if (node_matrix[i][k] == -1)
                                found = TRUE;
                        else
                        {
                                found = FALSE;
                                ++i;
                        }
                }

                if (!found)
                {
                        node_matrix[ifpos][k] = -1;
                        node_matrix[k][ifpos] = 1;
                }
         }

}

/***********************************************************************

generate_matrix_of_precedence(tnode **list, int n)

tnode *list
            The root children list;
int n
            The length of the root children list;

The major function for determine the strictly precedence relationships
among the
nodes in the rule-tree.

***********************************************************************/

generate_matrix_of_precedence(tnode **list, int n)
{

      int i,j,k,found,ifpos, exist_user;
        tnode *lvalue_node;

      i = 0;
      while ((list[i] != NULL)&&(list[i]->ntype != N_IF))
            ++i;

      if (list[i] != NULL)
            ifpos = i;      /* exist IF node */
      else
            ifpos = -1;     /* no IF node, pure actions like conversion
rule */

        i = 0;
        while (list[i] != NULL)
        {
            if (list[i]->valuetype == ASSIGN)
            {

                  lvalue_node = (list[i]->children)[0];


                  if (lvalue_node->ntype == N_TEMP)  /* temparary
variable generate during serialization */
                  {

                     determine_precedence_for_temp_variable(lvalue_node,
list, i);

                  }
                  else if ((lvalue_node->ntype == N_ITEM)||(lvalue_node-
>ntype == N_RUNTIME_ITEM))
                  {

                     backward_determine_precedence_for_item(lvalue_node,
i, ifpos, list);

                     forward_determine_precedence_for_item(lvalue_node,
i, n, list);
                  }
            }

            ++i;

          }/* end of while */


          if (ifpos == -1)
            return;        /* no IF node, so we do not need to consider
it */


         determine_precedence_for_ifnode_and_action(ifpos, n);
}


/*******************************************************************
serialize(root, O_node, O_pos)

tnode *root
            The root node;
tnode *O_node
            The direct child of the root;
int *O_pos
            The position of this node in the root children list;

The major function for the rule serialization.
*******************************************************************/

serialize(root, O_node, O_pos)
tnode *root;
tnode *O_node;
int *O_pos;
{

      if (O_node->valuetype == ASSIGN)
            serialize_assign (root, O_node, O_pos);
      else
            serialize_operation (root, O_node, O_pos);

}


/***********************************************************************

serialize_operation (root, O_node, O_pos)

tnode *root
                The root node;
tnode *O_node
                The direct child of the root;
int *O_pos
                The position of this node in the root children list;

The major function for the serialization of the operation node(subtree).

***********************************************************************/

serialize_operation (root, O_node, O_pos)
tnode *root;
tnode *O_node;
int *O_pos;
{
        int n,i,o_pos;
        tnode *o_node, *Ti_node, *Pi_node;

        n = O_node->nb_children;

        if (n == 0) return;

        for (i = 0; i < n; ++i)
        {
                Pi_node = (O_node->children)[i];

                if (
                    (card(Pi_node->sysnames) > 1)
                    ||((Pi_node->executed_in != NULL)
                        &&(O_node->executed_in !=NULL)
                        &&strcmp(Pi_node->executed_in, O_node->
executed_in))
                    ||((Pi_node->executed_in != NULL)
                        &&(O_node->executed_in ==NULL))
                    ||(((Pi_node->sysnames)[0] != NULL)&&(O_node->
executed_in != NULL)&&strcmp((Pi_node->sysnames)[0], O_node->
executed_in))
```

```
                    ||(((Pi_node->sysnames)[0] != NULL)&&(O_node->
executed_in ==NULL))
                   )
                {

                        getout_rewrite (root, O_node, *O_pos, Pi_node,
i, &o_node,&Ti_node);

                        if ((Pi_node->executed_in == NULL)&&
                            (belongto(O_node->executed_in, Pi_node->
sysnames)))
                                o_node->executed_in = O_node->
executed_in;
                        else
                                o_node->executed_in = NULL;

                        o_node->sysnames = Pi_node->sysnames;
                        Ti_node->executed_in = NULL;
                        o_pos = *O_pos;

                        serialize_assign (root, o_node,&o_pos);
                        *O_pos = o_pos + 1;

                        if ((O_node->executed_in == NULL)
                            ||((o_node->executed_in != NULL)
                            &&(!strcmp(o_node->executed_in, O_node->
executed_in)))
                            ||((Pi_node->executed_in != NULL)
                            &&!strcmp(Pi_node->executed_in, O_node->
executed_in)))
                        {
                                put_in_rewrite(root, o_node, o_pos,
O_node, i);
                                *O_pos = o_pos;
                                if (((O_node->children)[i]->executed_in)
!= NULL)
                                O_node->executed_in = ((O_node->
children)[i]->executed_in);
                        }
                }
        }

        if (O_node->executed_in == NULL)
        {
                O_node->sysnames = (char **)new_list();
        }
        else
        {
                O_node->sysnames = (char **)new_list();
                O_node->sysnames = (char **)add_elem(0, (tptr *)O_node->
sysnames, O_node->executed_in);
        }

}
```

```
/***********************************************************************

            Filename: part.c

            Description: Partitioning the rule-tree nodes so that
                         all nodes in each partition will be executed
                         in the same application system.

***********************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include "message.h"

#define DELETE 2
#define MAX_NODE 50

extern int node_matrix[MAX_NODE][MAX_NODE]; /* for all nodes in a rule
tree */

int matrix[MAX_NODE][MAX_NODE]; /* for operations in each partition */
int Matrix[MAX_NODE][MAX_NODE]; /* for partition process */
int temp_matrix[MAX_NODE][MAX_NODE]; /* for partition process */

extern tpart** partition;   /* the data structure for storing partition
information */


/***********************************************************************

add_unique_elem(int item, int ***list)

int item
            The integer which is added into the integer list;
int ***list
            The list containing integer elements;

Add a integer element into the list, if this element is already in the
list, we will not add this element into it.

***********************************************************************/

add_unique_elem(int item, int ***list)
{

    int i = 0;
    int *temp;

    temp = (int *)malloc(sizeof(int));
    *temp = item;

    while (((*list)[i] != NULL)&&(*((*list)[i]) != item))
          ++i;
    if ((*list)[i] == NULL)
    {
        (*list) = (int **)realloc((*list), sizeof(int *)*(i+2));
        (*list)[i] = temp;
        (*list)[i+1] = NULL;
    }

}


/***********************************************************************
delete_elem (int pos, tptr *list)

int pos
            The position of the element to be deleted;
tptr *list
            The list the deleted element belong to.

Delete the element at position pos from the list.

***********************************************************************/

delete_elem (int pos, tptr *list)
{
   int i;

   i  = pos;

   while (list[i] != NULL)
   {
        list[i] = list[i+1];
        ++i;
   }
}


/***********************************************************************
int included(tptr elem, tptr **list)

tptr elem
            The element being checked;
tptr **list
            The list being checked.

Check whether the element is in the list, return TRUE or FALSE.

***********************************************************************/

int included(tptr elem, tptr **list)
{
      int i;

      i = 0;
      while ((list[i] != NULL)&&(elem != list[i]))
```

```
            ++i;

      if (list[i] == NULL)
            return FALSE;
      else
            return TRUE;

}


/***********************************************************************
listAppend(list1, list2)

tptr **list1
            The list being appended;
tptr *list2
            The list will append on list1;

Append list2 onto list1.
***********************************************************************/

listAppend(list1, list2)
tptr **list1;
tptr *list2;
{
    int i = 0;
    int k,j;

    k = 0;
    while ((*list1)[k] != NULL)
          ++k;

    i = 0;
    while (list2[i] != NULL)
          {
           (*list1) = add_elem(k, (tptr*)(*list1), (tptr)list2[i]);
           ++k;
           ++i;
          }
}

/***********************************************************************
Generate_initial_partitions(F,root,node_num)

tpart ***F
            The list of partitions;
tnode *root
            The root node of rule-tree;
int node_num
            Total number of nodes under root node;

Generate the initial partitions (assign each node under root into a
partition).
***********************************************************************/
```

```
Generate_initial_partitions(F,root,node_num)
tpart ***F;
tnode *root;
int node_num;
{
      int i,j,m;
      tpart *elem;
      int *id;

        for (i = 0; i < node_num; i++)
        {

                elem = (tpart *)malloc(sizeof(tpart));
                id = (int *)malloc(sizeof(int));
                (*id) = i;
                elem->ids = (int **)new_list();
                elem->ids = (int **)add_elem(0,(tptr *)elem->ids,
(tptr)id);

                if ((root->children)[i]->executed_in != NULL)
                {
                elem->apname = (char *)malloc(strlen((root->
children)[i]->executed_in)+1);
                strcpy(elem->apname, (root->children)[i]->executed_in);
                }
                else if ((root->children)[i]->applname != NULL)
                {
                elem->apname = (char *)malloc(strlen((root->
children)[i]->applname)+1);
                strcpy(elem->apname, (root->children)[i]->applname);
                }
                else
                        elem->apname = NULL;

                elem->index = i;

                *F = (tpart **)add_elem(i, (tptr *)*F, (tptr)elem);

                for (j=0; j < node_num; j++)
                {
                        Matrix[i][j] = node_matrix[i][j];
                }
        }

}



/*************************************************************************
partition_rule(tnode *root, int node_num)

tnode *root
            The root node of the rule-tree;
int node_num
            The total number of nodes in the rule-tree to participate in
            the partitioning.

This is the major function for rule-tree partitioning.

************************************************************************/

partition_rule(tnode *root, int node_num)
{

      tpart **F;
      tpart *elem;
      int *id;
      int i,j,m;

      F = (tpart **)new_list();

      Generate_initial_partitions(&F,root,node_num);

      printf("before sort and optimize:\n\n");
      Mprint(node_num);
      Fprint(F);

      Sort_partition(F);

      printf("after sort :\n\n");
      Mprint(node_num);
      Fprint(F);

      Optimize_partition(F);

      printf("after optimize :\n\n");
      Mprint(node_num);
      Fprint(F);

      i=0;
      while (F[i] != NULL)
      {
            printf("\n\n\nThe below is partition %d:\n\n", i);

            Generate_ordered_list_of_operation(root, F[i]->ids);

            ++i;
      }

      partition = F;  /* assign to global variable */

}


/************************************************************************
Generate_ordered_list_of_operation(tnode *root, int **nodelist)

tnode *root
            The root of the rule-tree;
int **nodelist
```

```
            The list of indexes of the nodes in a partition.

For the partition F containing operations O1,..,On, generate the ordered
list of this operations and maintain its original precedence
constraints.
***********************************************************************/

Generate_ordered_list_of_operation(tnode *root, int **nodelist)
{
     tid **P, **R, **A, **p, **r;
     int i,j,m,c,k,k1,k2,temp;
     tid *temptr;
     int found;
     tid **Flist,**flist;
     tid *elem;


     Flist = (tid **)new_list();

     P = (tid **)new_list();
     R = (tid **)new_list();
     A = (tid **)new_list();

     i = 0;
     while (nodelist[i] != NULL)
     {
        elem =  (tid *)malloc(sizeof(tid));

        elem->id = *(nodelist[i]);
        elem->index = i;

        Flist = (tid **)add_elem(i, (tptr *)Flist, (tptr)elem);

        ++i;
     }

     printf("\nthe initial input value of operations of a partition:");
     FIDprint(Flist);


     m = i;

     for (i=0; i<m; i++)
     for (j=0; j<m; j++)
     {
        matrix[i][j] = node_matrix[Flist[i]->id][Flist[j]->id];
     }


     k1 = 0;
     k2 = 0;
     for (i = 0; i < m; ++i)
     {
        found = FALSE;
```

```
   j = 0;
   while((!found)&&(j < m))
   {

        found = (matrix[i][j] == -1);
        ++j;
   }

   if ((!found)&&(((root->children)[Flist[i]->id])->ntype != N_IF))
   {
       P = (tid **)add_elem(k1, (tptr *)P, (tptr)Flist[i]);
       ++k1;
   }
   else
   {
       R = (tid **)add_elem(k2, (tptr *)R, (tptr)Flist[i]);
       ++k2;
   }

}

c = m -1;

flist = (tid **)new_list();

i = 0;
while (Flist[i] != NULL)
{

   flist = (tid **)add_elem(i, (tptr *)flist, (tptr)Flist[i]);
   ++i;
}

while (P[0] != NULL)
{
   listAppend(&A, P);
   p = (tid **)new_list();
   r = (tid **)new_list();


   i = 0;
   while (P[i] != NULL)
   {

      if (P[i] != Flist[c])
      {
           for (k=0; k<m; k++)
           {
                   temp = matrix[P[i]->index][k];
                   matrix[P[i]->index][k] = matrix[c][k];
                   matrix[c][k] = temp;

                   temp = matrix[k][P[i]->index];
                   matrix[k][P[i]->index] = matrix[k][c];
```

```
                        matrix[k][c] = temp;
                }

                temp = P[i]->index;

                temptr = P[i];
                Flist[temp] = Flist[c];
                Flist[c] = temptr;

                Flist[c]->index = c;
                Flist[temp]->index = temp;

                /* so each P list's element current pos in Flist */

/*
                printf("test matrix change\n");
                mprint(m);
                FIDprint(Flist);
*/
            }
            --c;
            ++i;
        }

        k1 = 0;
        k2 = 0;

        i = 0;
        while (R[i] != NULL)
        {

                if ((matrix[c][R[i]->index] != 1)&&
(Found_all_operations(R[i]->index,A,Flist,m))&&((root->
children)[Flist[c]->id]->ntype != N_IF))
                {
                     p = (tid **)add_elem(k1, (tptr *)p, (tptr)R[i]);
                     ++k1;
                }
                else
                {
                     r = (tid **)add_elem(k2, (tptr *)r, (tptr)R[i]);
                     ++k2;
                }
                ++i;
        }


        P = p;
        R = r;

     }

     printf("\nthe matrix for the operations in this partition:\n");
     mprint(m);
     printf("\n the order of the operations in this partition:");
     FIDprint(Flist);

}


/***********************************************************************
mprint(int m)

int m
      The total number of nodes in a partition.

Print out the precedence matrix for all the nodes in a partition.

***********************************************************************/

mprint(int m)
{
        int i,j;

        for (i=0; i<m; i++)
        {
        for (j=0; j<m; j++)
        {
         printf("%d ",matrix[i][j]);
        }
        printf("\n");
        }
}


/***********************************************************************
Mprint(int node_num)

int node_num
            The total number of partitions.

Print out the precedenec matrix for partitions.

***********************************************************************/

Mprint(int node_num)
{
      int i,j,m;

        m = node_num;
      for (i=0; i<m; i++)
      {
      for (j=0; j<m; j++)
      {
       printf("%d ",Matrix[i][j]);
      }
      printf("\n");
      }
}
```

```
/***********************************************************************
FIDprint(tid **F)

tid **F
            The list of elements which contain the id for each node.

Print out the node ids in the F set.
***********************************************************************/

FIDprint(tid **F)
{
        int i,j;


                j = 0;
                while (F[j] != NULL)
                {
                        printf(" %d ", F[j]->id);
                        ++j;
                }
                printf("\n");

}


/***********************************************************************
int Found_all_operations(index, A, Flist, m)

int index
                Index for which operation node this function work on;
tid **A
                A list of operation nodes for checking;
tid **Flist
                The total operation nodes of the rule-tree;
int m
                The size of the node precedence matrix;

Check whether the operations that must be executed after the node
(specified by the index) F are in list A or not. If yes return TRUE,
otherwise returen FALSE.
***********************************************************************/

int Found_all_operations(index, A, Flist, m)
int index;
tid **A;
tid **Flist;
int m;
{
        int all_used;
        int i;

        all_used = TRUE;
        i = 1;
        while ((i < m)&& (all_used == TRUE))
        {
                if ((matrix[index][i] == -1)&&(included(Flist[i],A) ==
FALSE))
                     all_used = FALSE;
                ++i;
        }

        return (all_used);
}


/*********************************************************************
Fprint(tpart **F)

tpart **F
            The partition list of the rule-tree;

Print out the nodes containing in each partitions.

*********************************************************************/

Fprint(tpart **F)
{
      int i,j;

        i = 0;
        while (F[i] != NULL)
        {
                printf(" Flist[%d] set include: ", i);
                j = 0;
                while ((F[i]->ids)[j] != NULL)
                {
                        printf(" %d ", *((F[i]->ids)[j]));
                        ++j;
                }
                printf("\n");
                ++i;
        }

}


/*********************************************************************
Sort_partition(tpart **Flist)

tpart **Flist
            The partitions list of the rule-tree;

Place the partitions in a linear sequence and maintaining the original
precedence dependencies.
*********************************************************************/

Sort_partition(tpart **Flist)
```

```
{
     tpart **P, **R, **A, **p, **r;
     int i,j,m,c,k,k1,k2,temp;
     tpart *temptr;
     int found;
     tpart **flist;

     P = (tpart **)new_list();
     R = (tpart **)new_list();
     A = (tpart **)new_list();

     i = 0;
     while (Flist[i] != NULL)
          ++i;

     m = i;
     k1 = 0;
     k2 = 0;
     for (i = 0; i < m; ++i)
     {
        found = FALSE;

        j = 0;
        while((!found)&&(j < m))
        {
             found = (Matrix[i][j] == -1);
             ++j;
        }

        if (!found)
        {
            P = (tpart **)add_elem(k1, (tptr *)P, (tptr)Flist[i]);
          ++k1;
        }
        else
        {
            R = (tpart **)add_elem(k2, (tptr *)R, (tptr)Flist[i]);
            ++k2;
        }

     }

     c = m -1;

     flist = (tpart **)new_list();

     i = 0;
     while (Flist[i] != NULL)
     {
      flist = (tpart **)add_elem(i, (tptr *)flist, (tptr)Flist[i]);
        ++i;
     }

     while (P[0] != NULL)
     {
        listAppend(&A, P);
        p = (tpart **)new_list();
        r = (tpart **)new_list();


      i = 0;
      while (P[i] != NULL)
      {
           if (P[i] != Flist[c])
           {
            for (k=0; k<m; k++)
            {
                  temp = Matrix[P[i]->index][k];
                  Matrix[P[i]->index][k] = Matrix[c][k];
                  Matrix[c][k] = temp;

                  temp = Matrix[k][P[i]->index];
                  Matrix[k][P[i]->index] = Matrix[k][c];
                  Matrix[k][c] = temp;
            }

            temp = P[i]->index;

            temptr = P[i];
            Flist[temp] = Flist[c];
            Flist[c] = temptr;

            Flist[c]->index = c;
            Flist[temp]->index = temp;

            /* so each P list's element current pos in Flist */
/*
            printf("test matrix change\n");
            Mprint(node_num);
            Fprint(Flist);
*/
            }
          --c;
          ++i;
      }

      k1 = 0;
      k2 = 0;
      i = 0;
      while (R[i] != NULL)
      {
            if ((Matrix[c][R[i]->index] !=
1)&&(Found_all_partitions(R[i]->index,A,Flist,m)))
            {
                 p = (tpart **)add_elem(k1, (tptr *)p, (tptr)R[i]);
                 ++k1;
            }
            else
            {
                 r = (tpart **)add_elem(k2, (tptr *)r, (tptr)R[i]);
                 ++k2;
            }
            ++i;
      }


      P = p;
      R = r;

     }

}


/***********************************************************************
int Found_all_partitions(index, A, Flist, m)

int index
            Index for which partition this function work on;
tpart **A
            A list of partitions for checking;
tpart **Flist
            The total partitions of the rule-tree;
int m
            The size of the partitioning matrix;

Check whether the partitions that must be executed after partition
(specified by the index) F are in list A or not. If yes return TRUE,
otherwise returen FALSE.
***********************************************************************/

int Found_all_partitions(index, A, Flist, m)
int index;
tpart **A;
tpart **Flist;
int m;
{
      int all_used;
        int i;

      all_used = TRUE;

      i = 1;
      while ((i < m)&& (all_used == TRUE))
      {
            if ((Matrix[index][i] == -1)&&(included(Flist[i],A) ==
FALSE))
                 all_used = FALSE;
            ++i;
      }

      return (all_used);
}
```

```
/***********************************************************************
Optimize_partition(tpart **Flist)

tpart **Flist
            The list of partitions;

Try to reduce the total number of partitions to get the optimal
partitions, the optimal partitions will still be stored in
Flist.(according to Gilber's paper p.119 algorithm for optimal
partitioning.
**********************************************************************/

Optimize_partition(tpart **Flist)
{
      int change, size, m,n, nb_delete, i, j, k;

      i = 0;
      while (Flist[i] != NULL)
             ++i;

      m = i;

      change = TRUE;

      while (change == TRUE)
      {

            change = FALSE;

            size = m;

            for (i = m-1; i > 0; i--)
            {

                  if ((Flist[i]->apname == NULL)||
                      (Flist[i-1]->apname == NULL)||
                      ((Flist[i-1]->apname != NULL)&&(Flist[i]->apname
!= NULL)&&!strcmp(Flist[i-1]->apname, Flist[i]->apname)))
/*
                  if ((Flist[i]->apname == NULL)||
                           ((Flist[i-1]->apname !=
NULL)&&!strcmp(Flist[i-1]->apname, Flist[i]->apname)))
*/                {

                  listAppend(&(Flist[i-1]->ids), Flist[i]->ids);

                  if (Flist[i-1]->apname == NULL)
                  {
                        CopyStr(Flist[i-1]->apname, Flist[i]->apname);
                  }

                  Matrix[i-1][i] = 0;
                  Matrix[i][i-1] = 0;

                  for (j = 0; j < m; j++)
                  {
                        if (Matrix[i][j] == 1)
                        {
                              Matrix[i][j] = 0;
                              Matrix[j][i] = 0;
                              Matrix[i-1][j] = 1;
                              Matrix[j][i-1] = -1;
                        }
                        else if (Matrix[i][j] == -1)
                        {
                              Matrix[i][j] = 0;
                                        Matrix[j][i] = 0;
                                        Matrix[i-1][j] = -1;
                                        Matrix[j][i-1] = 1;
                                }
                  }

                  Matrix[i][i] = DELETE;
                  --m;
                  change = TRUE;
                  }
            }


            if (change == TRUE)
            {
            nb_delete = 0;
              for (i=0; i < size; i++)
            {
                  if (Matrix[i][i] == DELETE)
                  {
                        delete_elem(i-nb_delete, Flist);
                        ++nb_delete;
                  }
            }


            m = 0; n = 0;
            for (i=0; i < size; i++)
            {
                  if (Matrix[i][i] != DELETE)
                  {
                        for (j = 0; j < size; j++)
                        {
                              if (Matrix[j][j] != DELETE)
                              {
                                  temp_matrix[m][n] = Matrix[i][j];
                                  ++n;
                              }
                        }
                        ++m;
                  }
            }

            for (i=0; i < m; i++)
```

```
                for (j=0; j < n; j++)
                {
                        Matrix[i][j] = temp_matrix[i][j];
                }

            }

         }
}
```

# References

[Babin92] Babin Gilbert "Adaptiveness in Information System Integration" Ph.D. Thesis, Rensselaer Polytechnic Institute, August 1992

[Babin93] Babin Gilbert "Rule Oriented Programming Environment: Technical Specifications", Rensselaer Polytechnic Institute, August 1993

[Bouziane91] Bouziane M'hamed "Metadata Modeling and Management", Ph.D. Thesis, Computer Science, Rensselaer Polytechnic Institute, June 1991

[Cheung 91] Cheung, Weiman "The Model-Assisted Global Query System", Ph.D. Thesis Decision Science and Engineering Systems, Rensselaer Polytechnic Institute, November 1991

[Hsu92] Hsu Cheng, Babin Gilbert, Bouziane M'hamed, Cheung Waiman, Yee Lester "Metadadabase Modeling for Enterprise Information Integration", Journal of Systems Integration, 1,5-37,1992

[Oracle92] "Oracle 7 Server Concepts Manual", Oracle Corporation, 1992