## Appendix A. Module Specifications

The formal specification of each module contains seven primary sections.

1. Purpose

A natural language statement defining the purpose of the module is given. Such a statement adds clarity to the specifications.

2. Variable Declarations

In this section each variable used in the specification is declared and its "type" given. Because of the abstract nature of modules at higher levels, types other than the typical integer, real, boolean, and char are frequently used.

3. Constants

This section defines all constants used in the module specification.

4. Function Definitions

This section defines the operations of the module. The definition of each function includes a prototype call indicating how the function is referenced, a natural language statement indicating the purpose of the operation, and a list of exception conditions for which the function cannot be successfully invoked. The actual parameters supplied to the exception conditions may be the formal parameters of the prototype call or any of the module constants.

Each function may be of type ST, SV, or SE. An SV function returns a value directly. That is, a value is associated with its name. An SE function returns a value or values indirectly through its parameter list.

For each ST function an EFFECTS section axiomatically defines the state change of the machine caused by successful execution of the function. Each effect is a formal statement which defines the values of an SV or SE function of the module after the call in terms of the module constants, formal parameters of the prototype call, and values of the SV or SE function before the call. Each effect may be classified as local or non-local, depending upon whether the changes being defined are for a function of the module or for a function of an external module.

For each SV and SE function initial values are stated so that the initial module state may be determined.

5. Exception Condition Definitions

This section defines the conditions under which the module functions cannot be successfully invoked. Each exception condition is specified as a name with a formal parameter list and is defined axiomatically using SV and SE functions and the module constants. Exception conditions corresponding to non-local conditions are denoted by an asterisk. The specification of each exception condition also includes a natural language statement explaining its purpose. This adds clarity to the specification and can aid the implementor in determining the appropriate response to an occurence of the condition.

6. External Functions

Functions of other modules which are referenced in the specification are listed in this section.

7. Miscellaneous Definitions

This section defines certain abstractions which are used frequently

in the specification. The use of such abstractions can significantly reduce the length of the specifications and improve their readability and understandability.

## RELATION MODULE

### Purpose

This module defines the syntax and semantics of the operations which may be used to create and maintain relations of n-tuples.

### Variable Declarations

integer : i,j,n

boolean : b

tuple : r,s,t

relation : R,S,T

relational operator : $\theta$

domain name : $d, d_1, d_2, d_n, d_i$

user identification number : id,uid

relation type : rt

op : operation type

### Constants

relops : $\{ =, \neq, >, \geq, <, \leq \}$

opset : {'VIEW','MODIFY','INSERT','GRANT','AUTH','REVOKE','INT',
'DROP','SAVE','COMP','CREATE'}

### Function Definitions

1. R : = select(S,d,$\theta$,v)

   Type : ST, SV

   Purpose: Creates a new relation R which contains a subset of the

tuples of S based on their values in a particular domain.

Effects

local: relation_exists(R) = true;

ndom(R) = ndom(S);

$\forall i(1 \leq i \leq \text{ndom}(S))[\text{domain}(i,R) = \text{domain}(i,S)]$;

$r \subseteq R \Leftrightarrow r \subseteq \{s \mid s \subseteq S \wedge \text{compval}(s[d,S],\theta,v) = \text{true}\}$;

$\forall i(1 \leq i \leq \text{ndom}(S))[\text{domain\#}(\text{domain}(i,R),R) = \text{domain\#}(\text{domain}(i,S),S)]$;

rtype(R) = 'TEMP';

$\text{ntup}(R) = \text{card}\{s \mid s \subseteq S \wedge \text{compval}(s[d,S],\theta,v) = \text{true}\}$;

non-local: $\forall op(op \in \text{opset})[\text{check\_auth}(\text{uid},R,op) = \text{true}]$

Exception conditions: NO_RELATION(S)

RELATION_DEFINED(R)

BAD_DOMAIN(d,S)

BAD_RELOP(θ)

BAD_VTYPE(v,d,R)

NO_AUTH(uid,S,'VIEW')

BAD_RELCOMP(d,S,θ)

2. $R := \text{project}(S,d_1,d_2,\ldots,d_n)$

Type: ST, SV

Purpose: Creates a new relation, R, which is the projection of relation S over the domains specified by D.

Effects

local: relation_exists(R) = true;

ndom(R) = n;

$\forall i(1 \leq i \leq n)\ [\text{domain}(i,R) = d_i]$;

rtype(R) = 'TEMP';

$\forall_{i(1 \leq i \leq n)}$[domain#($d_i$,R) = i]

ntup(R) = {proj(s,S,$d_1$,$d_2$,...,$d_n$)| s $\in$ S};

r $\in$ R $\Leftrightarrow$ r $\in$ {proj(s,S,$d_1$,$d_2$,...,$d_n$)| s $\in$ S};

non-local : $\forall_{op(op \in opset)}$[check_auth(uid,R,op) = true]

Exception conditions: RELATION_DEFINED(R)

NO_RELATION(S)

BAD_DOMAIN($d_1$,R), BAD_DOMAIN($d_2$,R),...,

BAD_DOMAIN($d_n$,R)

NO_AUTH(uid,S,'VIEW')

3. R : = join(S,$d_1$,$\theta$,T,$d_2$)

Type: ST, SV

Purpose: Creates a new relation, R, which is the $\theta$-join of relation S and T over domains $d_1$ and $d_2$ respectively.

Effects

local: relation_exists(R) = true;

ndom(R) = ndom(S) + ndom(T);

r $\in$ R $\Leftrightarrow$ r $\in$ {s^t| s $\in$ S, t $\in$ T};

$\forall_{i(1 \leq i \leq ndom(S))}$[domain(i,R)=domain(i,S)];

$\forall_{i(ndom(S) < i \leq ndom(T)+ndom(S))}$

[domain(i,R) = domain(i,T)];

rtype(R) = 'TEMP'

$\forall_{i(1 \leq i \leq ndom(S))}$[domain(i,S)=d $\Rightarrow$ domain#(d,R)=i];

$\forall_{i(1 \leq i \leq ndom(S))}$[domain(i,T)=d $\Rightarrow$ domain#($d_1$R) = i+ndom(S)

ntup(R) = card({s^t| s $\in$ S, t $\in$ T})

non-local : $\forall_{op(op \in opset)}$[check_auth(uid,R,op) = true];

Exception conditions: RELATION_DEFINED(R)

NO_RELATION(T)

NO_RELATION(S)

BAD_DOMAIN($d_1$,S)

BAD_DOMAIN($d_2$,T)

BAD_RELOP($\theta$)

NO_AUTH(uid,S,'VIEW')

BAD_DTYPES($d_1$,S,$d_2$,T)

BAD_RELCOMP($d_1$,S,$\theta$)

BAD_RELCOMP($d_2$,T,$\theta$)

4. R : = restrict(S,$d_1$,$d_2$,$\theta$)

Type: ST, SV

Purpose: Creates a new relation, R, which is the $\theta$ - restriction of relation S over domains $d_1$ and $d_2$.

Effects

local: relation_exists(R) = true;

ndom(R) = ndom(S);

$r \underset{\sim}{\in} R \Leftrightarrow r \underset{\sim}{\in} \{s \underset{\sim}{\in} S \mid compval(s[d_1,S],\theta,s[d_2,S]) = \underline{true}\}$;

$ntup(R) = card(\{s \underset{\sim}{\in} S \mid compval(s[d_1,S],\theta,s[d_2,S]=\underline{true})\})$;

$\forall i(1 \leq i \leq ndom(S))[domain(i,R) = domain(i,S)]$;

$\forall i(1 \leq i \leq ndom(S))[domain(i,S) = d \Rightarrow domain\#(d,R)=i]$;

rtype(R) = 'TEMP';

non-local: $\forall op(op \in opset)[check\_auth(uid,R,op) = \underline{true}]$

Exception conditions: RELATION_DEFINED(R)

NO_RELATION(S)

BAD_DOMAIN($d_1$,S)

BAD_DOMAIN($d_2$,S)

BAD_RELOP(θ)

BAD_DTYPES($d_1$,S,$d_2$,S)

NO_AUTH(uid,S,'VIEW')

BAD_RELCOMP($d_1$,S,θ)

BAD_RELCOMP($d_2$,S,θ)

5. insert_tuple(r,R)

   Type: ST

   Purpose: Inserts a tuple into a relation.

   Effects: {'tuple_exists'(r,R) = false}=>{ntup(R)='ntup'(R)+1};

   tuple_exists(r,R) = true

   Exception conditions: NO_RELATION(R)

   NO_TUPLE(r)

   BAD_TSIZE(r,R)

   BAD_TUPLE(r,R)

   BAD_TVAL(r,R)

   NO_AUTH(uid,R,'INSERT')

6. create_relation(R,$d_1$,$d_2$,...,$d_n$)

   Type: ST, SE

   Purpose: Creates an empty relation with domains $d_1,d_2,\ldots,d_n$.

   Effects

   local: relation_exists(R) = true;

   ndom(R) = n;

   rtype(R) = 'TEMP';

   $\forall i(1 \leq i \leq n)$ [domain(i,R) = $d_i$];

   ntup(R) = 0;

$\forall i(1 \leq i \leq n)[domain\#(d_i,R) = i];$

non-local: $\forall op(op \in opset)[check\_auth(uid,R,op) = \underline{true}];$

Exception conditions: RELATION_DEFINED(R)

NO_AUTH(uid,,'CREATE')

7. n : = ntup(R)

Type: SV

Purpose: Returns the number of tuples contained in the relation.

Exception conditions: NO_RELATION(R)

NO_AUTH(uid,R,'VIEW')

Initial value: undefined

8. b : = relation_exists(R)

Type: SV

Purpose: Returns true if relation R is defined.

Exception conditions: NO_AUTH(uid,R,'VIEW')

Initial value: undefined

9. b : = tuple_exists(r,R)

Type: SV

Purpose: Returns true if the tuple r is contained in R.

Exception conditions: NO_RELATION(R)

BAD_TSIZE(r,R)

NO_TUPLE(r)

BAD_TUPLE(r,R)

BAD_TVAL(r,R)

NO_AUTH(uid,R,'VIEW')

Initial value: undefined

10. d : = domain(i,R)

Type: SV

Purpose: Returns the name of the ith domain of relation R.

Exception conditions: BAD_DNUM(i,R)

NO_RELATION(R)

NO_AUTH(uid,R,'VIEW')

Initial value: undefined

11. i : = domain#(d,R)

Type: SV

Purpose: Returns the position of domain d in relation R.

Exception conditions: NO_RELATION(R)

BAD_DOMAIN(d,R)

NO_AUTH(uid,R,'VIEW')

Initial value: undefined

12. n : = ndom(R)

Type: SV

Purpose: Returns the number of domains in a relation.

Exception conditions: NO_RELATION(R)

NO_AUTH(uid,R,'VIEW')

Initial value: undefined

13. drop_relation(R)

Type: ST

Purpose: Deletes relation R from the system.

Effects

local: relation_exists(R) = false

Exception conditions: NO_AUTH(uid,R,'DROP')

14. rt : = rtype(R)

Type: SV

Purpose: Returns the "type" of relation R.

Exception conditions: NO_AUTH(uid,R,'VIEW')

NO_RELATION(R)

Initial value: undefined

15. keep_relation(R)

Type: ST

Purpose: Saves the relation R

Effects

local: rtype(R) = 'BASE'

Exception conditions: NO_RELATION(R)

NO_AUTH(uid,,'SAVE')

Exception Condition Definitions

NO_RELATION(R): relation_exists(R) = false;

The user has specified an operation on an undefined relation.

RELATION_DEFINED(R): relation_exists(R) = true;

User has attempted to create a relation with the same name as an already existing relation.

BAD_DOMAIN(d,R): domain_exists(d,R) = false;

The user has specified a domain which does not exist for the specified relation.

BAD_RELOP(θ): θ $\notin$ relops

θ is not a relational operator.

BAD_RELCOMP(d,S,θ)*: check_relcomp(d,S,θ) = false

θ - comparisons are not allowed on domain d of relation S.

BAD_TSIZE(r,R): ncomp(r) ≠ ndom(R);

User has attempted to insert a tuple having an incorrect number of components into a relation.

NO_TUPLE(r): tuple_defined(r) = false;

User has specified a tuple which has not been defined.

BAD_TUPLE(r,R)*: ∃i(1 ≤ i ≤ ncomp(r))[check_type(d,R,r(i)) = false];

The data types of some component of tuple r does not match the data type of the corresponding domain of relation R.

BAD_VTYPE(v,d,R)*: var_type(v) ≠ domain_type(d,R);

User has specified a value for a relation domain which is not of the correct type.

BAD_DNUM(i,R): i ≤ 0 ∨ i > ndom(R)

User has specified an incorrect domain number.

BAD_TVAL(r,R)*: ∃ i(1 ≤ i ≤ ncomp(r)) [check_value(domain(i,R),R,r(i))=false];

User has attempted to insert a tuple containing a value not allowed by integrity assertions.

NO_AUTH(id,R,op)*: check_auth(id,R,op) = false

User has requested an operation which he does not have the authority to perform.

BAD_DTYPES($d_1$,R,$d_2$,S)*: check_dcomp($d_1$,R,$d_2$,S) = false;
User has attempted an operation to compare values of two incomparable domains.

External Functions

AUTHORIZATION: check_auth

INTEGRITY: domain_type, check_value, check_type
check_dcomp, check_relcomp

TUPLE: tuple_defined, ncomp

Miscellaneous Definitions

r[d,R]: r(domain#(d,R))

s $\in$ S : tuple_exists(s,S) = true

card(X): cardinality of set X

proj(s,S,$d_1$,$d_2$,...,$d_n$):(s[$d_1$,S],s[$d_2$,S],...,s[$d_n$,S])

r^s : (r(1),r(2),...,r(ncomp(r)),s(1),s(2),...,s(ncomp(s))

compval(v1,$\theta$,v2): returns true if the predicate v1 $\theta$ v2 is satisfied and false otherwise

domain_exists(d,R): $\exists$i(1 $\leq$ i $\leq$ ndom(R))[domain(i,R) = d]

uid: identification number of current user of the module

var_type(v): data type of the argument variable

## Tuple Module

Purpose

This module defines the syntax and semantics of the operations which may be used to create and modify n-tuples.

Variable Declarations

integer : n

boolean : b

tuple name : r

Function Definitions

1. define_tuple $(r,n,v_1,v_2,\ldots,v_n)$

   Type: ST, SV

   Purpose: Creates a tuple r of n components.

   Effects

   local: tuple_defined(r) = true:

   ncomp(r) = n;

   $\forall i(1 \leq i \leq n)[r(i) = v_i]$

   Exception conditions: none

2. b : = tuple_defined(r)

   Type: SV

   Purpose: Returns true if r is a defined tuple.

   Exception conditions: none

   Initial value: false

3. n : = ncomp(r)

   Type: SV

   Purpose: Returns the number of components of tuple r.

   Exception conditions: NO_TUPLE(r)

   Initial value: undefined

Exception conditions

NO_TUPLE(r): tuple_defined(r) = false

Miscellaneous Definitions

r(i) : $i^{th}$ component of tuple r.

## Authorization Module

Purpose

This module defines the syntax and semantics of operations which can be used to create and maintain system authorizations. Each authorization is an assertion which defines an allowable access for a system user.

Variable Declarations

| | |
|---|---|
| relation name: | R |
| domain name: | d |
| boolean: | b |
| user identification: | id |
| operation type: | op |

Constants

opset: {'VIEW','MODIFY','INSERT','GRANT','AUTH','REVOKE','INT', 'DROP','SAVE','COMP','CREATE'}

idset: set of authorized user numbers

Function Definitions

1. create_auth(id,R,op)

| | |
|---|---|
| Type: | ST |
| Purpose: | Creates the authorization for a user to perform an operation. |
| Effects | |
| local: | check_auth(id,R,op) = true |

Exception conditions: NO_RELATION(R)

BAD_ID(id)

BAD_OP(op)

NO_AUTH(uid,R,'AUTH')

2. revoke_auth(id,R,op)

Type: ST

Purpose: Revokes an authorization from a user.

Effects

local: check_auth(id,R,op) = false

Exception conditions: NO_RELATION(R)

BAD_ID(id)

BAD_OP(op)

NO_AUTH(uid,R,'REVOKE')

3. b := check_auth(id,R,op)

Type: SV

Purpose: Returns true if user has authorization to perform the operation.

Exception conditions: BAD_ID(id)

NO_RELATION(R)

BAD_OP(op)

Initial value: undefined

Exception Condition Definitions

NO_RELATION(R): relation_exists(R) = false

BAD_ID(id): check_id(id) = false

BAD_OP(op): op $\notin$ opset

NO_AUTH(id,R,op): check_auth(id,R,op) = false

External Functions

RELATION: relation_exists

Miscellaneous Definitions

check_id(id): id ∈ idset

uid:identification number of current user of module

## RNT Module

Purpose

This module defines the syntax and semantics of operations which can be used to create and maintain a table containing information about relation names.

Variable Declarations

boolean: b,inv

integer: i,j,k,ntup,ndom

relation name: R

relation type: rtype

domain table: dtable,dptr

user identification number: id

tuple table: data

Constants

rset = {'TEMP','BASE'}

Function Definitions

1. b:= get_rnval(R,rtype,ntup,ndom,dtable,data,id)

Type: SV, SE

Purpose: Returns entry values for a relation name.

Exception conditions: none

Initial values: rtype: undefined

ntup: undefined

ndom: undefined

dtable: undefined

data: undefined

id: undefined

2. set_rnval(R,rtype,ntup,ndom,dptr,data,id)

Type: ST

Purpose: Sets the entry values for a relation name.

Effects

local: get_rnval(R) = true;

$get\_rnval_2(R)$ = rtype;

$get\_rnval_3(R)$ = ntup;

$get\_rnval_4(R)$ = ndom;

$get\_rnval_5(R)$ = dptr;

$get\_rnval_6(R)$ = data;

$get\_rnval_7(R)$ = id

Exception conditions: none

3. b:=insert_rname(R)

Type: SV, ST

Purpose: Inserts a relation anem into the relation name table.

Effects

local: get_rnval(R) = true

Exception conditions: none

4. set_rtype(R,rtype)

Type: ST

Purpose: Defines the relation type for a relation name.

Effects

local: $get\_rnval_2(R)$ = rtype

Exception conditions: NO_RNAME(R)

BAD_RTYPE(rtype)

5. set_ntup(R,ntup)

Type: ST

Purpose: Defines the number of tuples for a relation name.

Effects

local: $get\_rnval_3(R)$ = ntup

Exception conditions: NO_RNAME(R)

6. set_ndom(R,ndom)

Type: ST

Purpose: Sets the number of domains for a relation name.

Effects

local: $get\_rnval_4(R)$ = ndom

Exception conditons: NO_RNAME(R)

7. set_dtable(R,dtable)

Type: ST

Purpose: Defines the domain table for a relation name.

Effects

local: $get\_rnval_5(R)$ = dtable

Exception conditions: NO_RNAME(R)

8. set_data(R,data)

Type: ST

Purpose: Defines the data table for a relation name.

Effects

local: $get\_rnval_6(R) = data$

Exception conditions: NO_RNAME(R)

9. create_rnt

Type: ST

Purpose: Creates an empty RNT table

Effects

local: rnt_defined = true

Exception conditions: RNT_EXISTS

10. b:= rnt_defined

Type: SV

Purpose: Returns true if the RNT table has been created.

Initial value: false

Exception conditions: none

11. set_id(R,id)

Type: ST

Purpose: Defines the owner of relation R.

Effects

local: $get\_rnval_7(R) = id$

Exception conditions: NO_RNAME(R)

BAD_ID(id)

Exception Condition Definitions

RNT_EXISTS: rnt_defined = true

NO_RNAME(R): get_rnval(R) = false

BAD_RTYPE(rtype): rtype $\notin$ rset

BAD_ID(id): id $\notin$ idset

Miscellaneous Definitions

idset: set of allowable user numbers

$get\_rnval_i(R)$ : the value of the $i^{th}$ return parameter of SE function 'get_rnval' with input parameter R.

## DNT Module

Purpose

This module defines the syntax and semantics of operations which can be used to create and maintain structures containing information about relation domains.

Variable Declarations

integer: n,nval

boolean: b,inv

domain name: d,dname

domain type: dtype

domain table: dtable,dptr

value table: vtable,v

Function Definitions

1. b:=get_dnval(dtable,dname,dtype,nval,vtable,inv)

Type: SV, SE

Purpose: Returns entry values for a domain name.

Exception conditions: none

Initial values: dtype: undefined

nval: undefined

ntable: undefined

inv: undefined

2. set_dnval(dtable,dname,dtype,nval,vptr,inv)

Type: ST

Purpose: Sets the entry values for a domain name.

Effects

local: get_dnval(dtable,dname) = true;

$get\_dnval_3$(dtable,dname) = nval;

$get\_dnval_4$(dtable,dname) = vtable;

$get\_dnval_5$(dtable,dname) = inv;

$get\_dnval_2$(dtable,dname) = dtype

Exception conditions: NO_DNAME(dname,dptr)

NO_DTABLE(dptr)

3. b:=check_dtable(dptr)

Type: SV

Purpose: Returns true if dptr is a defined domain table

Exception Conditions: none

Initial value: false

4. get_dname(i,dtable,dname,dtype,nval,vptr,inv)

Type: SE

Purpose: Returns the values of the ith entry in dtable.

Exception conditions: BAD_DNUMBER(i,dtable)

NO_DTABLE(dtable)

Initial values: dtype : undefined

nval : undefined

vptr : undefined

inv : undefined

5. i:= dname#(d,dtable)

Type: SV

Purpose: Returns the domain number of a domain in a domain table.

Exception conditions: NO_DNAME(d,dtable)

NO_DTABLE(dtable)

Initial value: undefined

6. i:=#dnames(dtable)

Type: SV

Purpose: Returns the number of domain names for a domain table.

Exception conditions: NO_DTABLE(dtable)

Initial value: undefined

7. dtable:=get_dtable(n)

Type: SV, ST

Purpose: Returns an empty domain table.

Exception conditions: none

Effects

local: check_dtable(dtable) = true;

#dnames(dtable) = 0

Exception Condition Definitions

NO_DNAME(d,dptr) : get_dnval(dptr,d) = false

NO_DTABLE(dptr): check_dtable(dptr) = false

BAD_DNUMBER(i,dptr) : $i \leq 0 \vee i > \#dnames(dptr)$

Miscellaneous Definitions

$get\_dnval_i$(dtable,dname) : the value of the $i^{th}$ return parameter of SE function 'get_dnval' with input parameters dtable and dname.

## Appendix B - Mapping Functions

The mapping function between two machines $M_i$ and $M_{i-1}$ is the means of defining how the states of $M_{i-1}$ are represented at the level of machine $M_i$. That is, given a state of $M_{i-1}$ it is possible to determine using the mapping function $f^i_{i-1}$ the corresponding state of $M_i$. Figure 7 illustrates the facts that 1) several states of $M_{i-1}$ may map to the same state of $M_i$ (This is due to delaying the binding of the state of $M_{i-1}$) and 2) a state of $M_{i-1}$ may have no image in the state set of $M_i$.

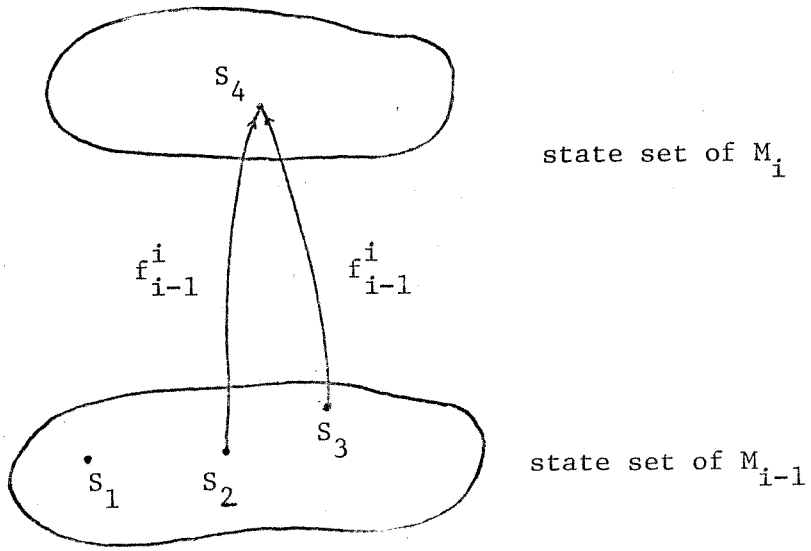



Figure 7. The mapping function between state sets of adjacent machines $M_i$ and $M_{i-1}$.

The mapping function $f^i_{i-1}$ is composed of a set of <u>partial mappings.</u> Each partial mapping of $f^i_{i-1}$ is an SV or SE function of $M_i$ set equal to

some expression containing the SV and SE functions of $M_{i-1}$. Below is the set of partial mapping funcitons which correspond to the RELAITON module of Figure s.

| | |
|---|---|
| relation_exists(R): | get_rnval(R) |
| domain(i,R): | get_dname(i,get_rnval$_6$(R)) |
| domain#(d,R): | dname#(d,get_rnval$_6$(R)) |
| ndom(R): | get_rnval$_4$(R) |

## Appendix C

We show here the derivation of several output assertions for the program (Appendix D) which implements the 'select' function of the RELATION module. This function has the prototype call:

$$R:=select(S,D,\theta,v)$$

The following assertions are obtained from the EFFECTS section of the formal specification of 'select':

$$relation\_exists(R) = \underline{true}$$

$$rdom(R) = ndom(S)$$

$$\forall i(1 \leq i \leq ndom(S)\,[domain(i,R) = domain(i,S)].$$

Moreover, the partial mapping functions of the SV functions of these assertions (Appendix B) are:

$$relation\_exists(R) : get\_rnval(R)$$

$$ndom(R) : get\_rnval_4(R)$$

$$domain(i,R) : get\_dname(i,get\_rnval_6(R))$$

The output assertions can then be derived by replacing each SV function occurrence in the EFFECTS assertion by its instantiated partial mapping function. The resulting output assertions are:

$$get\_rnval(R) = \underline{true}$$

$$get\_rnval_4(R) = get\_rnval_4(S)$$

$$\forall i(1 \leq i \leq get\_rnval_4(R))[get\_dname(i,get\_rnval_6(R)) = get\_dname(i,get\_rnval_6(S)].$$

## Appendix D

The following is an abstract program which implements the 'select' operation of the RELATION module. The exits correspond to the detection of exception conditions.

```
R:=select(s,d,θ,v)
begin
   if θ ∉ relops then exit1;
   b:=insert_rname(R);
   if b≠true
      then exit2
      else
         begin
            b:=get_rnval(S,rtype,ntup,ndom,dtable,data,id)
            if b≠true
               then exit3
               else
                  begin
                     b:=get_dnval(dtable,d,dtype,nval,ift,inv);
                     if b≠true
                        then exit4
                        else
                           begin
                              dt:=get_dtable(ndom);
                              for i:=1 to ndom do
                                 begin
```

```
                        get_dname(i,dtable,d,dtype);
                        set_dnval(dt,d,dtype);
                    end;
                end;
set_rtype(R,'TEMP');
set_ndom(R,ndom);
set_dtable(R,dt);
b:=find_value(ift,θ,v,x,n);
if inv=true
  then
    begin
    if b≠true
      then exit5;
      else
        begin
          for i:=1 to n do
            begin
              knt:=#tids(x(i));
              for j:=1 to knt do
                begin
                  tid:=get_tid(j,x(i));
                  insert_tid(tid,data);
                end;
              nt:=nt+knt;
            end;
        end;
```

```
            set_ntup(R,nt);
            set_data(R,data);
            end;
            else exit 6;
        end;
    end;
end;
```

## Appendix E

We provide in this section an example illustrating hierarchical performance evaluation as applied to the design of a machine which implements a single operation of a relational algebraic interface. The operation chosen is

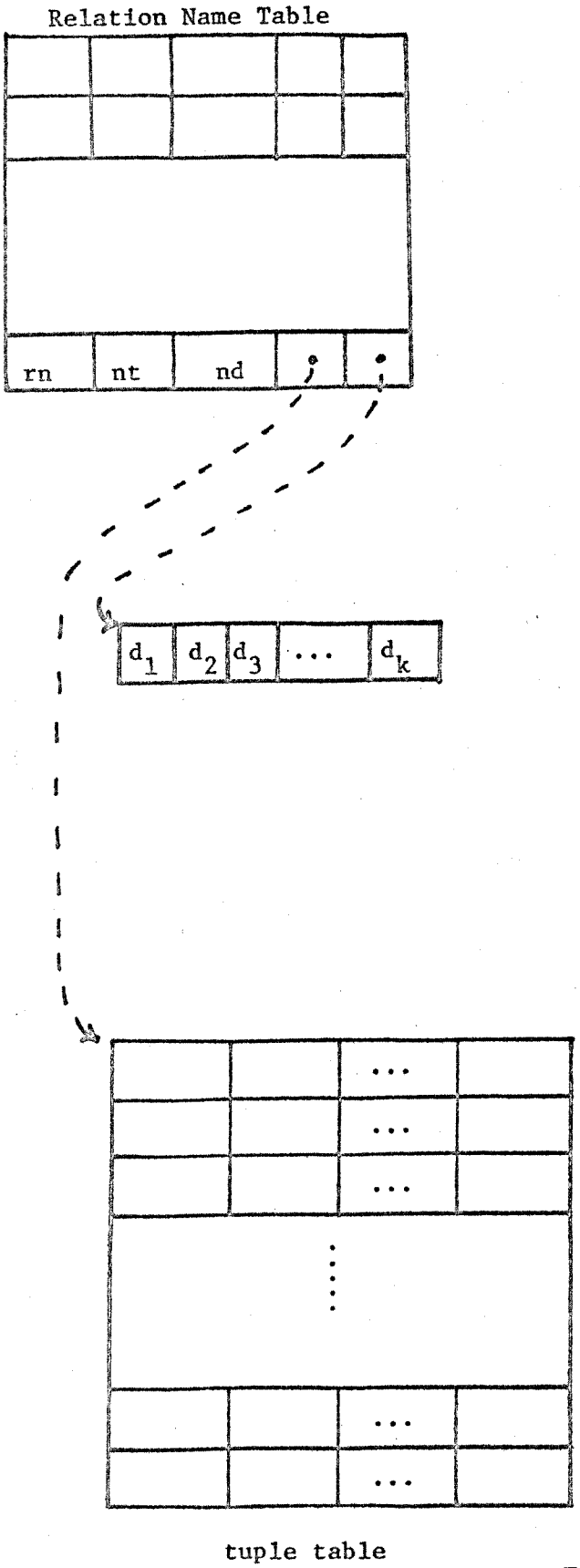
select (d,R,v)

which has the effect of printing all tuples of relation R whose values in domain d are equal to v. In this example we show two different machine designs which can be used to implement this operation.

Design 1 is shown in Figure 8. The design requires three abstract data objects and for each one the figure shows the operations used to access the structure and the design parameters. The implementation of the "select" operation using this design is shown in Figure 9. Performance assertions are used to express the expected execution time at various points in the program. These assertions can be verified by inspection. Finally in Figure 10 an analysis of the performance characteristics of the current design are shown. Figures 11 - 13 show the same process for the second design. This design is the same as that shown in with the exception that the "domain list" structure is replaced by the three structures of Figure 11.

It should be noted that the example presented here is a very crude attempt to analyze the tradeoffs involved in two alternative machine designs. The primary difficulties are that the designer's intuition is required in order to compare the performance characteristics of various abstract operations and that the analysis does not allow the designer to express his uncertainty about these judgments.

Design 1




get_rnval(i,rn,nt,nd,dptr,tptr)

$\bar{n}_R$: average number entries

$s_R$: entry size

$\bar{e}_R$: expected execution speed of get_rnval

get_dnval(i,dptr)

$\bar{n}_d$: average number of entries

$s_d$: entry size

$\bar{e}_d$: expected execution speed of get_dnval

get_tval(i,j,tptr)

$\bar{n}_t$: average number tuples

$\bar{s}_t$: average tuple size

$\bar{e}_t$: expected execution speed of get_tval

Figure 8

Miscellaneous Parameters

$c_1$: execution speed for simple assignment

$c_2$: execution speed for integer addition

$c_3$: execution speed for comparison

$c_4$: execution speed for printing a tuple.

Implementation of select (d,R,v)

```
begin
   i: = 0; j: = 0; k: = 0;
   repeat
      i: = i + 1;
      if  i > nr  then exit;
       get_rnval(i,rname,nt,nd,dptr,tptr);
   until  rname = R
```

$$\overline{E}_1 = 3c_1 + (2c_3 + c_1 + c_2 + \overline{e}_R)i$$

```
   repeat
      j: = j + 1;
      if j > n then exit;
       dname: = get_dnval(j,dptr);
   until  dname = d
```

$$\overline{E}_2 = \overline{E}_1 + (c_1 + c_2 + 2c_3 + \overline{e}_d)j$$

```
   repeat
         k: = k + 1;
         if get_tval(k,i,tptr) = v
              then  {print tuple k}
   until  k = nt
end.
```

$$\overline{E}_3 = \overline{E}_2 + (\overline{e}_t + c_2 + c_1 + 2c_3)nt + c_4$$

Figure 9

## Performance Analysis Design 1

$$\bar{E}_T = \bar{E}_3 = 3c_1 + (2c_3 + c_1 + c_2 + \bar{e}_R)i + (c_1 + c_2 + 2c_3 + \bar{e}_d)j + (\bar{e}_t + c_2 + c_1 + 2c_3)nt + c_4$$

$$= c_1 (3 + i + j + nt) + 2c_3 (i + j + nt) + c_4 + \bar{e}_R i + \bar{e}_d j + \bar{e}_t nt$$

Assumptions

$\bar{i} = 1/2\bar{n}_R$ $\quad \bar{nt} = \bar{n}_t$

$\bar{j} = 1/2\bar{n}_d$ $\quad \bar{e}_R \sim \bar{e}_d >> c_1 \sim c_2 \sim c_3 \sim c_4$

Case 1: $\bar{n}_R \sim \bar{n}_d \sim \bar{n}_t \quad \bar{e}_t \sim \bar{e}_R$

$$\bar{E}_T \sim 1/2\bar{n}_R\bar{e}_R + 1/2\bar{e}_d\bar{n}_d + \bar{e}_t\bar{n}_t$$

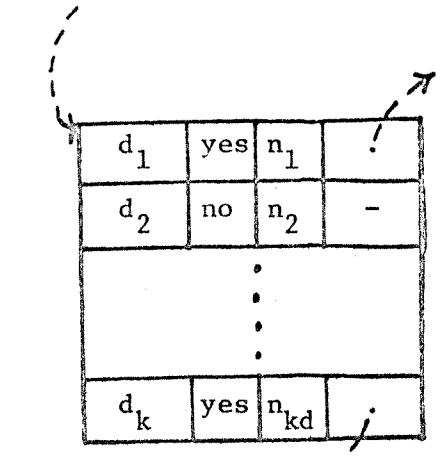
Case 2: $\bar{n}_R \sim \bar{n}_d << \bar{n}_t \quad \bar{e}_t >> \bar{e}_R$

$$\bar{E}_T \sim \bar{e}_t\bar{n}_t$$

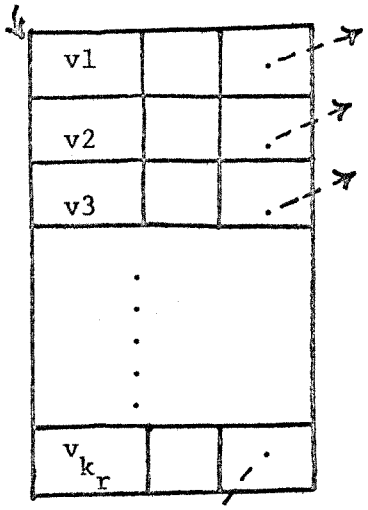
Storage Requirements

$$S = \bar{n}_R(s_R + \bar{n}_d s_d + \bar{n}_t \bar{s}_t)$$

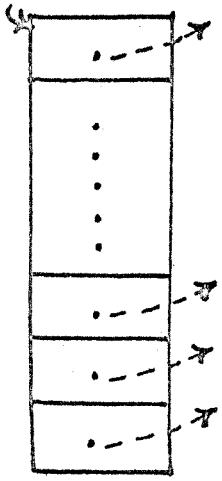
Figure 10




domain table




value table



location table

Design 2

get_dnval(i,dptr,dn,inv,nval,vptr)

$\bar{n}_d$: average number domains

$s_d'$: entry size

$\bar{e}_d'$: expected execution speed for get_dnval

get_val(i,vptr,val,nval,lptr)

$\bar{n}_v$: average number values

$s_v$: entry size

$\bar{e}_v$: expected execution speed for get_val

get_loc(i,lptr)

$\bar{n}_e$: average number of occurrences

$s_e$: entry size

$\bar{e}_e$: expected execution speed of get_loc

Figure 11

Implementation of SELECT(d,R,v) Design 2

```
begin
     i: = 0;  j: = 0;  k: = 0;
     repeat
        i: = i + 1;
        if  i > nr  then  exit;
         get_rnval(i,rname,nt,nd,dptr,tptr);
     until  rname = R
```

sequential search of relation name table

$$\bar{E}_1 = 3c_1 + (c_1 + c_2 + 2c_3 + \bar{e}_R)i$$

```
     repeat
        j: = j + 1;
        if  j > nd then  exit
        get_dnval(j,dptr,dname,inv,nval,vptr);
     until  dname = d
```

sequential search of domain table

$$\bar{E}_2 = \bar{E}_1 + (c_1 + c_2 + 2c_3 + \bar{e}_d')j$$

```
     if ¬inv
        then
           begin
                for  k: = 1  to  ntup  do
                   begin
                      if  get_tval(k,i,tptr) = v
                         then  {print tuple k}
                   end;
           end;
```

sequential search of tuple table

$$\bar{E}_3 = \bar{E}_2 + (\bar{e}_t + c_1 + c_2 + 2c_3)nt + c_4$$

Figure 12

```
else
   begin
      repeat
           k: = k + 1;
           if  k > nval          then  exit;
            get_val(k,vptr,val,lptr,knt);
      until   val = v
```

sequential search of value table

$$\bar{E}'_3 = E_2 + (\bar{e}_v + c_1 + c_2 + 2c_3)k$$

```
      for  m: = 1  to  knt  do
           begin
              s: = get_loc(m,lptr);
               {print tuple s}
           end
```

sequential search of location table

$$\bar{E}_4 = E'_3 + (2c_1 + c_2 + c_3 + \bar{e}_e)\ knt + c_4$$

```
      end;
end.
```

$$\bar{E}'_T = c_1(3 + i + j) + c_2(i + j) + 2c_3(i + j) + \bar{e}_R i + \bar{e}'_d j$$

$$+ \ \text{if } \neg \text{inv then } (\bar{e}_t + c_1 + c_2 + 2c_3)nt + c_4$$

$$\text{else } c_1(k + wknt) + c_2(k + knt) + c_3(2k + knt) + \bar{e}_v k + \bar{e}_e knt + c_4$$

Figure 12 (continued)

Performance Analysis - Design 2

Assumptions: $\bar{i} \sim 1/2\bar{n}_R \qquad \bar{k} \sim 1/2\bar{n}_v \qquad \bar{nt} \sim \bar{n}_t$

$\bar{j} \sim 1/2\bar{n}_d \qquad \bar{knt} \sim \bar{n}_e$

$\bar{e}_R \sim \bar{e}'_d \sim \bar{e}_v \sim \bar{e}_1 \gg c_1 \sim c_2 \sim c_3 \sim c_4$

Definitions: $\rho$ : probability that the domain specified in select is indexed. usage parameter

$\bar{\alpha}$ : average probability that any domain in the system is indexed. design parameter

$$\bar{E}_T \sim 1/2\bar{n}_R \bar{e}_R + 1/2\bar{n}_d \bar{e}_d' + \rho(1/2\bar{n}_v\bar{e}_v + \bar{n}_1\bar{e}_1) + (1-p)(\bar{n}_t\bar{e}_t)$$

Case 1: $\rho = 1$

$$\bar{E}_T \sim 1/2\bar{n}_R\bar{e}_R + 1/2\bar{n}_d\bar{e}_d' + 1/2\bar{n}_v\bar{e}_v + \bar{n}_1\bar{e}_1$$

If we assume $\bar{e}_R \ll \bar{e}_t$ and $\bar{n}_R \sim \bar{n}_d \sim \bar{n}_v \sim \bar{n}_1 \ll \bar{n}_t$ then $\bar{E}_T \ll \underbrace{\bar{n}_t\bar{e}_t}$

Case 2, Design 1

Storage requirements

$$S_2 = \bar{n}_R(S_R + \bar{n}_t \bar{S}_t + \bar{n}_d(S'_d + \bar{\alpha} \bar{n}_v(\bar{S}_v + \bar{n}_1 s_1)))$$

$$\sim S_1 + \bar{\alpha} \bar{n}_R \bar{n}_d \bar{n}_v(\bar{S}_v + \bar{n}_1 S_1)$$

Figure 13