The author gratefully acknowledges support from the taxpayers of Québec and of Canada who have paid his salary and research grants while this work was developed at McGill University, and from his students (who built the implementations and investigated the data structures and algorithms) and their funding agencies.

Computations, Procedures, and Events

Functions are special relations. Take this further:

- symmetric

- fully exploit relationship

e.g., $1 + I = (1 + i)^p$
(relate annual interest, I, to, say, monthly interest, i, if $p = 12$)

comp *IntPerChg*(I, i, p) **is**
    def $1 + I = (1 + i) ** p$
    $\{I <- (1 + i) ** p - 1\}$
    alt
    $\{i <- (1 + I) ** (1.0/p) - 1\}$
    alt
    $\{p <-$ round(log$(1 + I)$/log$(1 + i)$)$\}$
;

Computations

Computation = **comp**

= compressed relation

IntPerChg(	I	i	p)
	0	0	1
	0.1	0.1	1
	:	:	:
	0	0	2
	0.21	0.1	2
	:	:	:

So use relational algebra:

E.g., T-selector (select and project)

Intint <− $[p]$ **where** $I = 0.12$ & $i = 0.01$
in *IntPerChg*;

Intint(	p	)
	11	

(Syntactic sugar:)
Intint <− *IntPerChg*{0.12,0.01};

Computations

Symmetry

$$\textit{Intper} <\!- \; [i] \;\textbf{where}\; I = 0.12 \;\&\; p = 12 \;\textbf{in}\; \textit{IntPerChg};$$

Intper(	i	)
	0.00948882	

$$\textit{Intper} <\!- \; \textit{IntPerChg}\{0.12,,12\};$$

Another one

$$\textit{intper} <\!- \; [I] \;\textbf{where}\; i = 0.01 \;\&\; p = 12 \;\textbf{in}\; \textit{IntPerChg};$$

intper(	I	)
	0.126825	

$$\textit{intper} <\!- \; \textit{IntPerChg}\{,0.01,12\};$$

Computations

And why not invoke using a join?

IntPer(	I	p	)
	0.06	12	
	0.06	24	
	0.07	12	
	0.07	24	

ipc <– *IntPerChg* **natjoin** *IntPer*;

ipc(	I	p	i	)
	0.06	12	0.00486755	
	0.06	24	0.0024308	
	0.07	12	0.0056541	
	0.07	24	0.00282311	

(Syntactic sugar for relations, too:)

Ip <– *IntPer*{,12};

Ip(	I	)
	0.06	
	0.07	

Computations

Example 2.

comp *IntDiv*(*dividend, divisor, quotient, remainder*)
is
{ *quotient* <− *dividend*/*divisor* ;
remainder <− *dividend* **mod** *divisor*
};
divdiv <− *IntDiv*{34, 3, }; ≪ or ..{34, 3, , }
ddqr <− *IntDiv*{34, 3, 11, 1};
relation *DD*(*dividend, divisor*) <− {(34,3),(34,4)};
IDDD <− *IntDiv* **ijoin** *DD*;

divdiv(	*quotient*	*remainder*)
	11	1

ddqr(	*.bool*)
	`true`

IDDD(	*dividend*	*divisor*	*quotient*	*remainder*)
	34	3	11	1
	34	4	8	2

Computations

Example 3.

comp *SqRoot*(*sqr*, *root*) **is**
{ *root* <− sqrt(*sqr*)
  also
  root <− −sqrt(*sqr*)
};
sr <− *SqRoot*{24};

sr(	*root*	)
	-4.89898	
	4.89898	

Recursive Computations

comp *gcd*(k, m, g) **is**
$\{ \; g$ <− **if** $k < 0$ **then** *gcd*$\{-k, m\}$ **else**
if $m < k$ **then** *gcd*$\{m, k\}$ **else**
if $k = 0$ **then** m **else** *gcd*$\{m$ **mod** $k, k\}$
};

relation *km*(k, m) <− {
```
(0,2),(2,4),(4,6),(6,4),(13,27),(182,240),
(180,240),(-9,6),(9,-6),(-9,-6),(-6,-9)
```
};

kmg <− *km* **ijoin** *gcd*;

Recursive Computations gcd

$kmg(k$	m	$g)$
0	2	2
2	4	2
4	6	2
6	4	2
13	27	1
182	240	2
180	240	60
-9	6	3
9	-6	3
-9	-6	3
-6	-9	3

Procedures

- Top-level computations.

- Relational/domain algebra statements (including if-then[-else] and (recursive) procedure calls).

- Actual parameters passed by name, and specified **in** or **out** at call time.

comp *ABCjoindecomp*(R, S, T) **is**
{ T <−R **ijoin** S; }
alt
{ R <−$[A, B]$ **in** T;
S <−$[B, C]$ **in** T;
};

relation $R(A, B)$ <− {(0,0),(2,0),(0,1),(1,1)};
relation $S(B, C)$ <− {(0,0),(1,1),(0,1),(1,2)};
ABCjoindecomp(**in** R, **in** S, **out** T);
ABCjoindecomp(**out** U, **out** V, **in** T);

Procedures

- Attribute parameters are **in** or **out**

comp *joindecomp*(X, Y, Z, R, S, T) **is**
{ $T <- R$ **ijoin** S; }
alt
{ $R <- [X, Y]$ **in** T;
$S <- [Y, Z]$ **in** T;
};

joindecomp(**out** A, **out** B, **out** C,
in U, **in** V, **out** W);

joindecomp(**in** A, **in** B, **in** C,
out X, **out** Y, **in** W);

N.B. **comp** is called **proc** in current implementation

Recursive Procedures

```
comp Closure(TC, Graph, Parent, Child) is
{ temp <- Graph;
  TC <- Graph;
  Closure(in TC, in Graph, in Parent, in Child);
} alt
{ temp <- temp[Child icomp Parent]Graph;
  if [] in temp then
  { TC <- TC ujoin temp;
    Closure(in TC, in Graph, in Parent, in Child)
  };
};                                   << Rebecca Lui, 1996 >>

relation Parent(Sr, Jr) <- {
 ("Joe", "Tom"), ("Joe", "Mary"), ("Joe", "Pete"),
 ("Tom", "Kim"), ("Tom", "Sue"), ("Pete", "Sam")
};

Closure(out Ancestor, in Parent, in Sr, in Jr);
```

Event Programming

An event is a system-generated procedure call.

An event handler is a procedure.

relation *Inventory*
(*PartNo, Descr, QOH, Supplier, Thr, ROQ*) <−
{
```
        ("1", "widget", 23, "Acme", 20, 50),
        ("2", "gizmo", 97, "Zedco", 10, 30)
```
};

Event Programming

update *Inventory* **change**
QOH <− **if** *PartNo*=`"1"` **then** *QOH* −9 **else** *QOH*;

Inventory

(*PartNo*	*Descr*	*QOH*	*Supplier*	*Thr*	*ROQ*)
1	widget	14	Acme	20	50
2	gizmo	97	Zedco	10	30

SupplyHist

(*PartNo*	*Descr*	*Supplier*	*Ordr*	*Rcvd*	*Date*)
1	widget	Acme	50	DC	981103

Event Programming

update *Inventory* **change**
QOH<−**if** *PartNo*=`"1"` **then** *QOH*+45 **else** *QOH*;

Inventory

(*PartNo*	*Descr*	*QOH*	*Supplier*	*Thr*	*ROQ*)
1	widget	59	Acme	20	50
2	gizmo	97	Zedco	10	30

SupplyHist

(*PartNo*	*Descr*	*Supplier*	*Ordr*	*Rcvd*	*Date*)
1	widget	Acme	50	DC	981103
1	widget	Acme	DC	45	981105

Event Programming

```
comp post:change:Inventory[QOH]() is
{ let Date be "981103"; << today!
  let oldQOH be QOH;
  let Ordr be ROQ;
  let Rcvd be (long dc);
  Reorder <- [PartNo,Descr,Supplier,Ordr,Rcvd,Date]
    where QOH < oldQOH in (
     (where QOH ≤ Thr and Thr < oldQOH in Inventory)
     ijoin [PartNo,oldQOH] in TRIGGER
    );
  let Date be "981105"; << today!
  let Ordr be (long dc);
  let Rcvd be QOH - oldQOH;
  Resupply <- [PartNo,Descr,Supplier,Ordr,Rcvd,Date]
    where QOH > oldQOH in (
     ([PartNo,Descr,Supplier,QOH] in Inventory)
     ijoin [PartNo,oldQOH] in TRIGGER
    );
  SupplyHist <+ Reorder;   SupplyHist <+ Resupply;
};
```

Event Programming: TRIGGER

```
comp reset() is
{ relation Inventory
  (PartNo, Descr, QOH, Supplier, Thr, ROQ) <-
  { ("1", "widget", 23, "Acme", 20, 50),
    ("2", "gizmo", 97, "Zedco", 10, 30)
};};
comp pre:change:Inventory[QOH]() is
{ pr!!TRIGGER };
comp post:change:Inventory[QOH]() is
{ pr!!TRIGGER };

reset();
update Inventory change
QOH <- if PartNo="1" then QOH-9 else QOH;
```

TRIGGER .. both times

(*PartNo*	*Descr*	*QOH*	*Supplier*	*Thr*	*ROQ*)
1	widget	23	Acme	20	50
2	gizmo	97	Zedco	10	30

Event Programming: TRIGGER

relation *Widget*(*PartNo*, *newQOH*) <−
{ ("1", 13)};
reset();
update *Inventory* **change** *QOH* <− *newQOH*
using *Widget*;

TRIGGER		.. both times				
(*PartNo*	*Descr*	*QOH*	*Supplier*	*Thr*	*ROQ*	*newQOH*)
1	widget	23	Acme	20	50	13

comp pre:delete:*Inventory*() **is** { **pr!!** *TRIGGER* };
comp post:delete:*Inventory*() **is** {**pr!!** *TRIGGER*};
update *Inventory* **delete** *Widget*;

TRIGGER		.. both times			
(*PartNo*	*Descr*	*QOH*	*Supplier*	*Thr*	*ROQ*)
1	widget	23	Acme	20	50
2	gizmo	97	Zedco	10	30

Event Programming: TRIGGER

comp pre:add:*Inventory*() **is** { **pr!!** *TRIGGER* };
comp post:add:*Inventory*() **is** { **pr!!** *TRIGGER* };
relation *Doohickey*
(*PartNo, Descr, QOH, Supplier, Thr, ROQ*) <–
{ ("3", "doohickey", 50, "Fuller", 30, 50) };
reset();
update *Inventory* **add** *Doohickey*;

TRIGGER .. both times

(*PartNo*	*Descr*	*QOH*	*Supplier*	*Thr*	*ROQ*)
3	doohickey	50	Fuller	30	50

Event Programming: housekeeping

Note that we have created two event handlers of the form

comp pre:change:*Inventory*[*QOH*]() **is**

They coexist, so the *event name*,
pre:change:*Inventory*[*QOH*],
is ambiguous.

So **pr!!** and **sr!!** do not work with the event names, and the implementation presently only lets us use the system-generated tags (e.g., 8_526 and 6_621).

pr!!8_526

However, **eventoff** and **eventon** do use the event names, and deactivate or reactivate, respectively, all event handlers with the same name.

eventoff pre:change:*Inventory*[*QOH*];

(**dr!!** does not work.)

Event Programming: cascades

Event handlers can do further updates, so events can cascade.

The following is an extreme case.

```
domain n intg;
relation start(n) <- {(4)};
comp post:add:iota() is
{ let nmin1 be (red min of n) - 1;
  let n be nmin1;
  update iota add
    [n] in [nmin1] where nmin1>= 0 in iota;
};
update iota add start;
```

Computations with State

comp *counter*(*ct*) **is**
  state *count* **intg**;
  { *count* <− *ct*
  } **alt**
  { *count* <− *count* + 1;
   ct <− *count*
  };

c0 <− *counter*{0};
c1 <− *counter*{};
c2 <− *counter*{};
c3 <− *counter*{};

c3(*ct*)
3

Computations with State: Abstraction and Instantiation

```
domain DEP, BAL intg;
domain DEPOSIT comp(DEP);
domain BALANCE comp(BAL);
comp ba(BALANCE, DEPOSIT) is
  state bal intg;
  state oldbal intg;
  { comp DEPOSIT(DEP) is
   { oldbal <- bal;
     bal <- bal + DEP;
   } alt
   { DEP <- bal - oldbal;    };
   comp BALANCE(BAL) is
   { BAL <- bal;    };
   bal <- 0;
  };

relation accts(ACCNO, CLIENT) <- {
  (1729,"pat"),(4104,"suz")
};
```

Computations: Abstraction and Instantiation

accounts <− *accects* **ijoin** *ba*; **instantiation!**

ACCNO	*CLIENT*	*bal*	*oldbal*
1729	pat	0	0
4104	suz	0	0

update *accounts* **change** *DEPOSIT*(100)
using where *ACCNO* = 4104 **in** *accounts*;

comp *transfer*(*FROMACC*, *TOACC*, *AMT*) **is**
{ **update** *accounts* **change** *DEPOSIT*(−*AMT*)
  using where *ACCNO*=*FROMACC* **in** *accounts*;
  update *accounts* **change** *DEPOSIT*(*AMT*)
  using where *ACCNO* = *TOACC* **in** *accounts*;
}

transfer(**in** 1729, **in** 4104, **in** 50);