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.

Semistructure from Relations

T. H. Merrett
McGill University

Part I

- *A programming language talk disguised as a database talk.*
- *Why relations?*
- *Why semistructure?*

Part II

- *Relations and path expressions.*

Part III

- *Irregular and unknown structure.*

Part IV

- *Markup and data on the web.*

Integration. Integration. Integration.

Dedicated to the late Alberto Oscar Mendelzon

Part I   A PL talk disguised as a DB talk

- Not top-down (e.g., functional, logic, constraint paradigms).

- But bottom-up: programming for secondary storage.

- But not, for the moment, about
  - performance or optimization (NB terabytes or more),
  - compiler flexibility or debugging.

Why relations?

- Unit for bulk data:
  the essence of SS programming
  (hence necessary for DB).

- High-level operations (relational algebra).

- Abstraction over looping:
  like LISP, APL, SETL, .. but for SS.

Why semistructure?

- Bottom-up approach is empirical, not theoretical.

- So language must be tested in many applications.

- Semistructured data is a DB hot topic, but sufficiently worked out to provide a good test.

What is semistructure?

- Characterized by irregular or unknown structure.

- Notable language feature is path expressions.

www.cs.mcgill.cs/~tim/semistruc/rel2semi.ps.gz
www.cs.mcgill.cs/~tim/semistruc/recnest.ps.gz

Part II Relations and path expressions

- Paths of attributes.

- Paths of conditions.

- Paths for updates.

Paths of attributes

- Operations on relations.
- Operations on attributes.
- Nested relations and level changes.
- Paths of attributes.

Paths of attributes: relations

Family tree example 1.

Child(*Name*	*DoB*	*Pa*	*Ma*)
Mary	1934	Ted	Alice
James	1935	Ted	Alice
Joe	1933	Max	Sal

Unary operators.

ChildND <− [*Name*, *DoB*] **where** *DoB* > 1933
in *Child*;

ChildND(*Name*	*DoB*)
Mary	1934
James	1935

Name **in** *Child* ≡ [*Name*] **in** *Child*

(*Name*)
Mary
James
Joe

Paths of attributes: relations, cont.

Family tree example 1.

Child(

Name	*DoB*	*Pa*	*Ma*
Mary	1934	Ted	Alice
James	1935	Ted	Alice
Joe	1933	Max	Sal

)

Spouse(

Ma	*Pa*	*Wed*
Alice	Ted	1933

)

Binary operators. Note: infixed syntax.
([*Ma*, *Pa*] **in** *Child*) **ujoin** [*Ma*, *Pa*] **in** *Spouse*

(

Ma	*Pa*
Alice	Ted
Sal	Max

)

djoin, **ijoin**, ..

Paths of attributes: attributes

Domain algebra: **red**, **equiv**

let *Oldest* **be red min of** *DoB*;

equiv min of *DoB* **by** *Pa*

equiv min of *DoB* **by** *Ma*, *Pa*

Family tree example 1.

Child (*Name*	*DoB*	*Pa*	*Ma*	)	*Oldest*	[**min** *Pa*]	[**min** *Ma*,*Pa*]
Mary	1934	Ted	Alice		1932	1932	1934
James	1935	Ted	Alice		1932	1932	1934
Joe	1933	Max	Sal		1932	1933	1933
Pete	1932	Ted	Sal		1932	1932	1932

Paths of attributes: nesting

Domain algebra subsumes relational algebra.
let *ChildN* **be** [*Name*] **in** *Children*;
famChildN <− [*ChildN*] **in** *Family*;
$\Rightarrow$ *famChildN*(*ChildN*(*Name*))

Level-raising through anonymous singleton
FamChildN <− [**red ujoin of** *ChildN*] **in** *Family*;
Family tree example 2.

Family (*Ma*	*Pa*	*Wed*	*Children* (*Name*	*DoB*))	*ChildN* (*Name*)	*famChildN* (*ChildN*) (*Name*)	*FamChildN* (*Name*)
Alice	Ted	1933	Mary	1934	Mary	Mary	Mary
			James	1935	James	James	James
							Pete
Sal	Ted	1930	Pete	1932	Pete	Pete	Mary
							James
							Pete

Path expression
Family/*ChildN* $\equiv$ [**red ujoin of** *ChildN*] **in** *Family*

Paths of attributes

Family tree example 3

Person									
(*Name*	*Family*								
	(*Conj*	*Wed*	*Children*						
			(*Name*	*DoB*	*Family*				
					(*Conj*	*Wed*	*Children*		
							(*Name*	*DoB*	*Family*)
Ted	Alice	1933	Mary	1934	Max	1956	Sue	1957	—
							Tom	1958	—
			James	1935	Ann	1959	Joe	1960	—
	Sal	1930	Pete	1932	—				

Person/*Family*/*Children*/*Name* ≡
[**red ujoin of**
[**red ujoin of** — Mary
[*Name*] **in** — James
Children] **in** — Pete
Family] **in**
Person

Paths of attributes (cont.)

(Family tree example 3.)

Option

Person(/*Family*/*Children*)?/*Name* ≡

Name **in** *Person* **ujoin**	
[**red ujoin of**	`Ted`
[**red ujoin of**	`Mary`
[*Name*] **in**	`James`
Children] **in**	`Pete`
Family] **in**	
Person	

Kleene Star (recursive domain algebra)

Person(/*Family*/*Children*)*/*Name* ≡

let *Nom* **be** *Name* **ujoin**	`Ted`
[**red ujoin of**	`Mary`
[**red ujoin of**	`James`
Nom] **in**	`Pete`
Children] **in**	`Sue`
Family;	`Tom`
[**red ujoin of** *Nom*] **in** *Person*	`Joe`

Paths of conditions

Family tree example 1.

Child(*Name*	*DoB*	*Pa*	*Ma*)
`Mary`	`1934`	`Ted`	`Alice`
`James`	`1935`	`Ted`	`Alice`
`Joe`	`1933`	`Max`	`Sal`

Nullary relation is Boolean

[ ] **in** *Child* ≡ "something in *Child*" ≡
"there is a child" **true**

[ ] **where** *Name* = `"Joe"` **in** *Child* **true**

Path expression
(Family tree example 3).

Name **where** *Family*/*Children*/*Name* = `"Mary"`
in *Person* ≡
Name **where**
([ ] **where**
([ ] **where** *Name* = `"Mary"` **in**
Children) **in**
Family) **in**
Person

Paths of conditions, cont.

Recursive path expression

Name **where** (*Family*/*Children*/)**Name* = `"Mary"`
    in *Person* ≡

func *mary* **is**
{ *Name* = `"Mary"` **or**
    ([ ] **where**
        ([ ] **where** *mary* **in** *Children*)
    in *Family*)
};
Name **where** *mary* **in** *Person*

NB **and**, **xor**, etc. have no syntactic sugar.

Paths for updates

Family tree example 1.

Child(*Name*	*DoB*	*Pa*	*Ma*)
Mary	1934	Ted	Alice
James	1935	Ted	Alice
Joe	1933	Max	Sal

update *Child* **change**
DoB <− **if** *Name* = `"Mary"` **then** `"1933"`
else *DoB*;

Child(*Name*	*DoB*	*Pa*	*Ma*)
Mary	1933	Ted	Alice
James	1935	Ted	Alice
Joe	1933	Max	Sal

Paths for updates, cont.

Path expression
(Family tree example 3).

update *Person/Family/Children* **change**
    DoB $<-$ **if** *Name* $=$ `"Mary"` **then** `"1933"`
        else *DoB*; $\equiv$

update *Person* **change**
    update *Family* **change**
        update *Children* **change**
            DoB $<-$ **if** *Name* $=$ `"Mary"` **then** `"1933"`
                else *DoB*;

Paths for updates, cont.

Recursive path expression

```
update Person(/Family/Children)* change
    DoB <- if Name = "Mary" then "1933"
                else DoB; ≡
proc mary33 is
{  DoB <- if Name = "Mary" then "1933"
              else DoB;
   if [ ] in Family then update Family change
      if [ ] in Children then
         update Children change mary33;
};
update Person change mary33;
```

Part III Irregular and unknown structure

- Schema query and update.
  Transpose metadata
  operator, originally devised
  for association data mining.

- Missing and multiple values.

- Wildcards.

- Schema discovery.

Schema query and update.

Union type

Family tree example 4.

domain *DoB* **strg**|**intg**;

Child(*Name*	*DoB*	*Pa*	*Ma*	)
Mary	**intg:**1934	Ted	Alice	
James	**strg:**1935	Ted	Alice	

Transpose operator

domain *att* **attr**;
domain *typ* **type**;
domain *val* **any**;
let *xpose* **be** **transpose**(*att, typ, val*);
transposeChild <−
  [*Name, DoB, Pa, Ma, xpose*] **in** *Child*;

transposeChild

(*Name*	*DoB*	*Pa*	*Ma*	*xpose*			)
				(*att*	*typ*	*val*	)
Mary	**intg:** 1934	Ted	Alice	Name	**strg**	**strg:**Mary	
				DoB	**intg**	**intg:**1934	
				Pa	**strg**	**strg:**Ted	
				Ma	**strg**	**strg:**Alice	
James	**strg:** 1935	Ted	Alice	Name	**strg**	**strg:**James	
				DoB	**strg**	**strg:**1935	
				Pa	**strg**	**strg:**Ted	
				Ma	**strg**	**strg:**Alice	

Schema query and update, cont.

Query on structure
Find all integer dates of birth

intgDoB <− **where** *xpose/att* = **quote** *DoB* **and**
xpose/typ = **intg in** *Child*;

intgDoB

(*Name*	*DoB*	*Pa*	*Ma*	)
Mary	**intg:**1934	Ted	Alice	

Update on structure
domain *DoB* **strg**|**intg**;

Child(*Name*	*DoB*	*Pa*	*Ma*	)
Mary	**intg:**1934	Ted	Alice	
James	**strg:**1935	Ted	Alice	

update *Child* **change** *DoB* <− (**strg**)*DoB*
using where *xpose/att* = **quote** *DoB* **and**
xpose/typ = **intg in** *Child*;

Child(*Name*	*DoB*	*Pa*	*Ma*	)
Mary	**strg:**1934	Ted	Alice	
James	**strg:**1935	Ted	Alice	

Missing and multiple values

I By union type
domain *child* **strg**;
domain *DoB* **intg**;
domain *Name* **strg**;
domain *chiln*(*Name, DoB*);
domain *Children child|chiln*;
domain *Conj* **strg**;
domain *Wed* **strg**;

Family(*Conj*	*Wed*	*Children*)
Alice	1933	*child*:Bernice

relation *Chiln*(*DoB,Name*) <−
{(1934,"Mary"),(1935,"James")};
update *Family*/*Children* **add** *Chiln* ≡
update *Family* **change**
update *Children* **add** *Chiln*;

Family(*Conj*	*Wed*	*Children*)
Alice	1933	*child*: Bernice
		chiln:
		(*Name* *DoB*)
		Mary 1934
		James 1935

Missing and multiple values, cont.

II By polymorphic relation
domain *Conj* **strg**;
domain *Wed* **strg**;
domain *Child* **strg**;
let *Name* **be** *Child*;
let *Children* **be relation**(*Name*);

Family(	*Conj*	*Wed*	*Child*	)	*Name*	*Children* (*Name*)
	Alice	1933	Bernice		Bernice	Bernice

update *Family* **change**
    replace *Child* **with** *Children*;
update *Family*/*Children* **add** *Chiln*

Family(	*Conj*	*Wed*	*Children* (*Name*)	)
	Alice	1933	Bernice (*DoB* *Name*) 1934 Mary 1935 James	

Wildcards

Family tree example 5.

FamEmp						
(*Name*	*Family*		*Employer*			)
	(*Conj*	*Wed*)	(*Boss*	*Conj*	*Subord*)	
`Ted`	`Alice`	`1933`	`Pete`	`Alan`	`Carole`	

famEmp/./Conj $\equiv$
  [**red ujoin of**
    [**red ujoin of** *Conj*] **in**
  .] **in** *FamEmp*

...should give `Alice, Alan`:

Wildcards, cont.

Transpose analyses leaves only:
transposeAll(*att*,*typ*) for non-leaf
as well as for leaf attributes.
let *nonleaves* **be transposeAll**(*att*) **djoin**
transpose(*att*);

FamEmp			*nonleaves*
(*Name*	*Family*	*Employer*)	(*att*)
	(..)	(..)	`Family`
			`Employer`

let *FE* **be** [**red ujoin of eval** *att*] **in** *nonleaves*;
famEmp/./*Conj* $\equiv$ *famEmp*/*FE*/*Conj* $\equiv$
[**red ujoin of**
[**red ujoin of** *Conj*] **in**
FE] **in** *FamEmp*

(*Conj*)
`Alice`
`Alan`

Recursion and wildcards

$$Person//Name \equiv Person(/.)^*/Name \equiv$$

let *Nom* **be** *Name* **ujoin**
  [**red ujoin of** *Nom*] **in** .;
[**red ujoin of** *Nom*] **in** *Person*;

Family tree example 3

Person									
(*Name*	*Family*								
	(*Conj*	*Wed*	*Children*						
			(*Name*	*DoB*	*Family*				
					(*Conj*	*Wed*	*Children*		
							(*Name*	*DoB*	*Family*))))))
Ted	Alice	1933	Mary	1934	Max	1956	Sue	1957	—
							Tom	1958	—
			James	1935	Ann	1959	Joe	1960	—
	Sal	1930	Pete	1932	—				

(*Name*):{(Ted), (Mary), (James), (Pete), (Sue), (Tom), (Joe)}

Schema discovery

```
Person
(Name   Family                              )
        (Conj   Children                    )
                (Name   Family         )
                        (Conj   Children)
                                (Name)
```

let *attrib* **be self**;
let *schema* **be transpose**(*attrib*) **union**
[*attrib*, *schema*] **in** .;
Schema <− [*attrib*, *schema*] **in** *Person*;

```
Schema
(attrib  schema                                      )
         (attrib  schema                             )
                  (attrib   schema                   )
                            (attrib   schema         )
                                      (attrib  schema)
                                               (attrib)
```

```
Person   Name
         Family   Conj
                  Children  Name
                            Family    Conj
                                      Children  Name
```

Part IV

Markup and data on the web.

Semstructure/text

- Specialized operator, **mu2nest**: marked-up $\rightarrow$ nest, including order information.

- Text querying: metadata relational operator, **grep**.

Other applications

- Data Streams? Skyline?

www.cs.mcgill.cs/∼tim/semistruc/rel2semi.ps.gz
www.cs.mcgill.cs/∼tim/semistruc/recnest.ps.gz

Highlights

- Binary operators must be infixed.
- Nullary relations are Boolean.
- Need domain algebra as well as relational algebra.
- Domain algebra subsumes relational algebra for nesting.
- Nesting => recursive nesting (no 2nd class).
- Metadata is important for advanced work:
  - **transpose**, **transposeAll**, **quote**, **eval**, **self**

Conclusion

- Thinking relations through carefully $\rightarrow$

- Subtle adjustments to query syntax $\rightarrow$

- General purpose SS programming language $\rightarrow$

- Everything XML and XQuery can do and more.

Integration. Integration. Integration.