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.

T. H. Merrett ©06/11

Semistructured data

1. I Path expressions
   (a) Paths of attributes
   (b) Paths of conditions
   (c) Paths for updates

2. II Irregular and Unknown structure
   (a) III Schema query and update
   (b) Missing and multiple values
   (c) Wildcards
   (d) Schema discovery

3. Markup and Data on the Web

Path expressions: paths of attributes

Family

(*Ma*	*Pa*	*Wed*	*Children* (*Name*	*DoB*))
Alice	Ted	1932	Mary	1934
			James	1935
Mary	Alex	1954	Joe	1956
Jane	James	1960	Tom	1961
			Sue	1962

FamChildren <– [**red ujoin of** *Children*] **in** *Family*;

FamChildren

(*Name*	*DoB*)
Mary	1934
James	1935
Joe	1956
Tom	1961
Sue	1962

Syntactic sugar: **path expression**

FamChildren <– *Family*/*Children*;

Paths of attributes (cont.)

It also works for virtual attributes:

let *ChildN* **be** [*Name*] **in** *Children*;

Family / *ChildN*
(*Name*)
`Mary`
`James`
`Joe`
`Tom`
`Sue`

Special consideration for leaves:

Family / *Ma*
(*Ma*)
`Alice`
`Mary`
`Jane`

? [**red ujoin of** *Ma*] **in** *Family* No!

! [**red ujoin of relation** (*Ma*)] **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

[*Name*] **in**

Children] **in**

Family] **in**

Person

Mary
James
Pete

Paths of attributes (cont.)

(Family tree example 3.)

Option

Person(/*Family*/*Children*)?/*Name* ≡
Name **in** *Person* **ujoin**
 [**red ujoin of**
  [**red ujoin of**
   [*Name*] **in**
   Children] **in**
  Family] **in**
Person

```
Ted
Mary
James
Pete
```

Kleene Star (recursive domain algebra)

Person(/*Family*/*Children*)*/*Name* ≡
let *Nom* **be** *Name* **ujoin**
  [**red ujoin of**
   [**red ujoin of**
   Nom] **in**
   Children] **in**
  Family;
 [**red ujoin of** *Nom*] **in** *Person*

```
Ted
Mary
James
Pete
Sue
Tom
Joe
```

Paths of conditions

(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 3).

update *Person*/*Family*/*Children* **change**
    DoB <− **if** *Name* = `"Mary"` **then** `"1933"`
            else *DoB*; ≡
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*;

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* ≡

[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

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

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

(*Conj*)
`Alice`
`Alan`

Recursion and wildcards

Person//*Name* ≡ *Person*(/.)*/*Name* ≡

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 III

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

- Multimedia?

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