

conexus

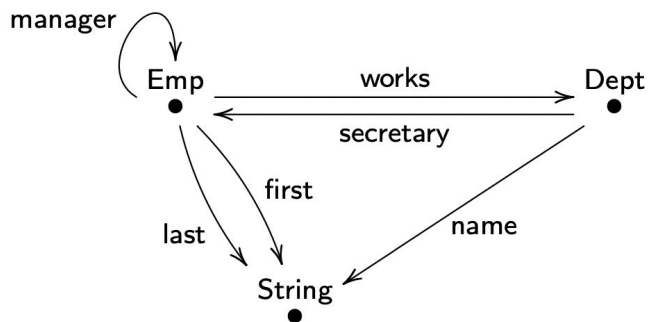
Summary

ACT 2022

Conexus Basics

- Spun out of MIT in 2015 to commercialize functorial data migration (David Spivak)
- Originally SBIR funded, now VC&revenue funded
- Clients ranging from Energy to Banking to Government
- Technical focus on the automated theorem proving and chase algorithms required to compute with co-presheaves at scale
- Developers of the CQL IDE (categoricaldata.net)
- Looking to both collaborate with academics and integrate data commercially (contact: ryan@conexus.com)

Co-presheafs / Ologs / Algebraic Databases



$$\text{Emp} \xrightarrow{\text{manager}} \text{Emp} \xrightarrow{\text{works}} \text{Dept} = \text{Emp} \xrightarrow{\text{works}} \text{Dept}$$

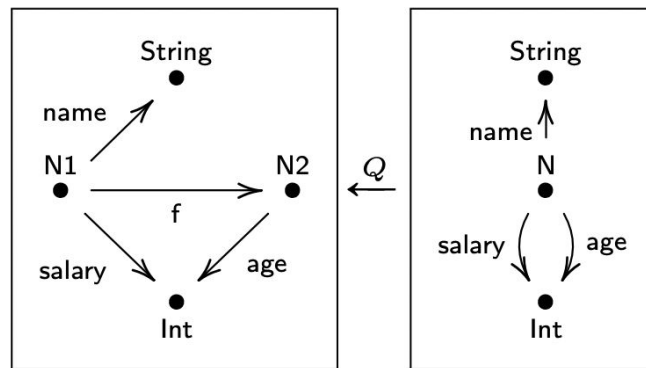
$$\text{Dept} \xrightarrow{\text{secretary}} \text{Emp} \xrightarrow{\text{works}} \text{Dept} = \text{Dept}$$

Emp				
ID	mgr	works	first	last
101	103	q10	Al	Akin
102	102	x02	Bob	Bo
103	103	q10	Carl	Cork

Dept		
ID	sec	name
q10	101	CS
x02	102	Math

String	
ID	
Al	
Bob	
...	

Functorial Data Migration



N1			
ID	name	salary	f
1	Alice	\$100	4
2	Bob	\$250	5
3	Sue	\$300	6

N2	
ID	age
4	20
5	20
6	30

$\xleftarrow{eval_Q}$
 $\xrightarrow{coeval_Q}$

N			
ID	name	salary	age
a	Alice	\$100	20
b	Bob	\$250	20
c	Sue	\$300	30

Data Integration

Observation			Person	Type
ID	f	g	ID	ID
			<i>p</i>	BP
				Wt

→

Method			Type
ID	g2		ID
<i>m</i> ₁	BP		BP
<i>m</i> ₂	BP		Wt
<i>m</i> ₃	Wt		
<i>m</i> ₄	Wt		

Observation			Person
ID	f	g1	ID
<i>o</i> ₁	Pete	<i>m</i> ₁	Jane
<i>o</i> ₂	Pete	<i>m</i> ₂	Pete
<i>o</i> ₃	Jane	<i>m</i> ₃	
<i>o</i> ₄	Jane	<i>m</i> ₁	

↓

Gender			Type
ID			ID
F			BP
M			Wt
			HR

Observation			Person	
ID	f	g	ID	h
<i>o</i> ₅	Peter	BP	Paul	M
<i>o</i> ₆	Paul	HR	Peter	M
<i>o</i> ₇	Peter	Wt		

→

Method		Observation		
ID	g2	ID	f	g1
<i>null</i> ₁	BP	<i>o</i> ₁	Peter	<i>m</i> ₁
<i>null</i> ₂	Wt	<i>o</i> ₂	Peter	<i>m</i> ₂
<i>null</i> ₃	HR	<i>o</i> ₃	Jane	<i>m</i> ₃
<i>m</i> ₁	BP	<i>o</i> ₄	Jane	<i>m</i> ₁
<i>m</i> ₂	BP	<i>o</i> ₅	Peter	<i>null</i> ₁
<i>m</i> ₃	Wt	<i>o</i> ₆	Paul	<i>null</i> ₂
<i>m</i> ₄	Wt	<i>o</i> ₇	Peter	<i>null</i> ₃

Gender		Type	Person	
ID		ID	ID	h
F		BP	Jane	<i>null</i> ₄
M		Wt	Paul	M
<i>null</i> ₄		HR	Peter	M

Copresheaves as Initial Algebras of Presentations

The screenshot displays the CQL IDE interface with a formal specification on the left and a summary of the resulting data on the right.

Formal Specification (Left Panel):

```
1 typeside TyJava = literal {
2 schema SJava = literal : TyJava {
3   entities
4     Employee
5     Department
6   foreign_keys
7     manager : Employee -> Employee
8     worksIn : Employee -> Department
9     secretary : Department -> Employee
10  path_equations
11    Employee.manager.worksIn = Employee.worksIn
12    Department.secretary.worksIn = Department
13    Employee.manager.manager = Employee.manager
14  attributes
15    first last : Employee -> string
16    age : Employee -> nat
17    cumulative_age : Employee -> nat
18    name : Department -> string
19  observation_equations
20    forall e. cumulative_age(e) = plus(age(e), age(manager(e)))
21  options
22    prover_simplify_max = 64
23 }
24 instance IJava = literal : SJava {
25 generators
26   a b c : Employee
27   m s : Department
28 equations
29   first(a) = Al
30   first(b) = Bob last(b) = Bo
31   first(c) = Carl
32   manager(c) = c
33   worksIn(a) = m worksIn(b) = m worksIn(c) = s
34   secretary(s) = c secretary(m) = b
35   secretary(worksIn(a)) = manager(a)
36   worksIn(a) = worksIn(manager(a))
37   age(a) = "2"
38   age(manager(a)) = "1"
39 options
40   prover_simplify_max = 64
41 }
42 #basic query syntax
```

Summary (Middle Panel):

- typeside TyJava
- schema SJava
- instance IJava : SJava
- query Q : SJava -> cod_q simple
- instance J : cod_q simple : SJava {
- schema SS
- instance I1 : SS
- typeside Ty
- schema S
- instance I : S

Department (2) Table:

Row	name	secretary
0	?0	3
1	?1	4

Employee (3) Table:

Row	age	cumulative_age	first	last	manager	worksIn
2	2	3	Al	?2	3	0
3	1	2	Bob	Bo	3	0
4	?3	(?3 plus ?3)	Carl	?4	4	1

Footer: 5 IDs, 5 nulls, 0.124 seconds. Provenance: Row limit:

Static Checking of (Pro) Functors

The screenshot shows the CQL IDE interface. The main editor contains the following code:

```
1 typeside Ty = literal {
10
11 schema Source = literal : Ty {
12   entities    Man Woman
13   attributes
14     fav_book_m : Man -> String    fav_book_w : Woman -> String
15     name_m    : Man -> String    name_w    : Woman -> String
16     paying   : Man -> Bool      }
17 schema Target = literal : Ty {
18   entities    Male GoodMatch PayingGoodMatch
19   foreign_keys
20     is_a : PayingGoodMatch -> GoodMatch
21     for_man : GoodMatch -> Male
22   attributes  man_name : Male -> String    woman_name : GoodMatch -> String }
23 query Q = literal : Source -> Target {
24   entity
25     GoodMatch -> {
26       from m:Man w:Woman
27       where fav_book_m(m) = fav_book_w(w)
28       attributes woman_name -> name_w(w)
29       foreign_keys for_man -> {man -> m}
30     }
31   entity
32     Male -> {
33       from man:Man
34       attributes man_name -> name_m(man)
35     }
36   entity
37     PayingGoodMatch -> {
38       from man:Man woman:Woman
39       where #fav_book_m(man) = fav_book_w(woman)
40             paying(man) = true
41       foreign_keys is_a -> {m -> man w -> woman}
42     }
43 }
```

The right-hand pane shows a proof tree:

```
Prove  err Sort 
> typeside Ty
> schema Source
> schema Target
> query Q : Source -> Target
```

The bottom pane contains the following diagnostic message:

In transform for foreign key is_a, Source instance equation `m.fav_book_m = w.fav_book_w` translates to `man.fav_book_m = woman.fav_book_w`, which is not provable in the target instance, displayed below. To proceed, consider removing it or adding more equations to the target instance.

```
generators
  man:Man woman:Woman
equations
  man.paying = true
```

Deciding Group Theory in CQL

The screenshot displays the CQL IDE interface. The main window is titled "CQL IDE" and contains a code editor on the left and a console at the bottom. The code editor shows a CQL script defining a group structure and solving an equation. Two dialog boxes are open, showing the results of solving equations in different contexts. The console window at the bottom provides performance metrics for the computation.

```
1 #one = two should not be provable
2 typeside EmptySortsCheck = literal {
14
15 typeside Group = literal {
16   types
17   S
18   constants
19   e : S
20   functions
21   I : S -> S
22   o : S,S -> S
23   equations
24   forall x. (e o x) = x
25   forall x. (I(x) o x) = e
26   forall x, y, z. ((x o y) o z) = (x o (y o z))
27   options
28   prover = completion
29 #   completion_precedence = "e o I"
30 }
31
32
```

Dialog 1 (KB - 6:17:48 PM):

- Buttons: DP, Model, Cayley, Text, Expression
- Buttons: Decide Equation-in-ctx, Show Info
- Input (either equation-in-ctx or term-in-ctx): forall x. (x o x) = e
- Output: false

Dialog 2 (KB - 6:17:28 PM):

- Buttons: DP, Model, Cayley, Text, Expression
- Buttons: Decide Equation-in-ctx, Show Info
- Input (either equation-in-ctx or term-in-ctx): forall x. (x o I(x)) = e
- Output: true

Console (KB - 6:17:48 PM):

```
Computation wall-clock time: 0.5s
GUI building time: 0.1s
typeside computation total time: 1.9s
JVM Used Change: 0 MB. Used Max: 221 MB.
```


Cayley Graph of Quaternions in CQL

The image shows a screenshot of a software application named "Cayley" running at 6:14:02 PM. The interface is divided into a code editor on the left and a graph visualization on the right. The code editor contains four schema definitions for different algebraic structures: Dihedral2, Quaternions, Cyclic4, and FreeMonoid. The Quaternions schema is the most detailed, defining entities, foreign keys, and path equations. The graph visualization shows a complex network of nodes and edges, with nodes labeled with expressions like $x.y_0.x.x_0$ and $y_0.x_0.y_0.x_0$. The edges are labeled with x and y , representing the generators of the group. A single edge is highlighted in cyan.

```
1 schema Dihedral2 = literal : empty {
2   entities
3   G
4   foreign_keys
5   r s : G -> G
6   path_equations
7   G.r.r = G
8   G.s.s = G
9   G.s.r.s = G.r
10 }
11
12 schema Quaternions = literal : empty {
13   entities
14   G
15   foreign_keys
16   x y x0 y0 : G -> G
17   path_equations
18   G.x.x0=G
19   G.x0.x=G
20   G.y.y0=G
21   G.y0.y=G
22   G.x.x.x.x = G
23   G.x.x = G.y.y
24   G.y0.x.y = G.x0
25 }
26
27 schema Cyclic4 = literal : empty {
28   entities
29   G
30   foreign_keys
31   f : G -> G
32   path_equations
33   G.f.f.f.f=G
34 }
35
36
37 schema FreeMonoid = literal : empty {
38   entities
39   G
40   foreign_keys
41   a b : G -> G
42 }
43
```

Cayley - 6:14:02 PM
Computation wall-clock time: 1.2s
GUI building time: 0.1s

JSON,RDF,XML etc import/export

The screenshot shows the CQL IDE interface. The main window is titled "CQL IDE" and has a menu bar with "Run", "New", "Open", "Save", "Deploy", "Options", and "Example: RDF Jena". Below the menu bar, there are several tabs, including "Untitled 1.cql". The main editor area contains a query with the following code:

```
1 schema S = literal : sql {
2
18
19 instance I = literal : S {
37
38 command exportRdfData = export_rdf_inst
42
43 instance J = import_rdf_all "file.xml"
44
45 instance K = spanify J
46
```

The summary pane on the right shows the following information:

```
Summary
schema S
instance I : S
command exportRdfData
instance J : literal : rdf { entities
instance K : schemaOf spanify J
```

The results pane on the right shows a table of results for the query. The table has three columns: "Row", "object", and "subject". The results are grouped by query path:

Row	object	subject
cql://attribute/Department/name (2)		
0	Math	cql://entity/Department/1
1	CS	cql://entity/Department/0
cql://attribute/Employee/age (3)		
2	2	cql://entity/Employee/0
3	1	cql://entity/Employee/1
4	2	cql://entity/Employee/2
cql://attribute/Employee/first (3)		
5	Carl	cql://entity/Employee/0
6	Bob	cql://entity/Employee/1
7	Al	cql://entity/Employee/2
cql://attribute/Employee/last (1)		
8	Bo	cql://entity/Employee/1
cql://foreign_key/Department/secretary (2)		
9	cql://entity/Employee/1	cql://entity/Department/1
10	cql://entity/Employee/0	cql://entity/Department/0
cql://foreign_key/Employee/manager (3)		
11	cql://entity/Employee/0	cql://entity/Employee/0
12	cql://entity/Employee/1	cql://entity/Employee/1
13	cql://entity/Employee/1	cql://entity/Employee/2
cql://foreign_key/Employee/worksIn (3)		
14	cql://entity/Department/0	cql://entity/Employee/0
15	cql://entity/Department/1	cql://entity/Employee/1
16	cql://entity/Department/1	cql://entity/Employee/2
http://www.w3.org/1999/02/22-rdf-syntax-ns#type (14)		
17	cql://entity/Employee/0	cql://entity/Employee/0

At the bottom of the results pane, it shows "45 IDs, 0 nulls, 0.01 seconds." and "Provenance: Row limit:

At the bottom of the IDE, there is a status bar showing "RDF Jena - 6:20:09 PM" and computation times:

```
Computation wall-clock time: 1.3s
GUI building time: 0.1s
schema computation total time: 0.1s
instance computation total time: 0.2s
command computation total time: 0.5s
```