

# A Practical Theory of Language Integrated Query

Philip Wadler, University of Edinburgh  
(joint work with James Cheney and Sam Lindley)

Data Science CDT

20 October 2016

# Databases vs. Programming Languages

“The problem with having two languages is ‘*impedance mismatch*’. One mismatch is conceptual—the data language and the programming language might support different paradigms. . . . The other mismatch is structural—the languages don’t support the same datatypes . . .”

—George Copeland and David Maier,  
*Making Smalltalk a Database System*,  
SIGMOD, 1984

“Databases and programming languages have developed almost independently of one another for the past 20 years.”

—Malcolm Atkinson and Peter Buneman,  
*Types and Persistence in Database Programming Languages*,  
Computing Surveys, 1987.

# Database programming languages

## Kleisli

Buneman, Libkin, Suciu, Tannen, Wong (Penn)

## Ferry

Grust, Mayr, Rittinger, Schreiber (Tübingen)

## Links

Cooper, Lindley, Wadler, Yallop (Edinburgh)

## SML#

Ohori, Ueno (Tohoku)

## Ur/Web

Chlipala (Harvard/MIT)

## LINQ for C#, VB, F#

Hejlsberg, Meijer, Syme (Microsoft Redmond & Cambridge)

# Flat data

departments

dpt
"Product"
"Quality"
"Research"
"Sales"

employees

dpt	emp
"Product"	"Alex"
"Product"	"Bert"
"Research"	"Cora"
"Research"	"Drew"
"Research"	"Edna"
"Sales"	"Fred"

tasks

emp	tsk
"Alex"	"build"
"Bert"	"build"
"Cora"	"abstract"
"Cora"	"build"
"Cora"	"design"
"Drew"	"abstract"
"Drew"	"design"
"Edna"	"abstract"
"Edna"	"call"
"Edna"	"design"
"Fred"	"call"

# Departments where every employee can abstract

```
select d.dpt as dpt
from departments as d
where not(exists(
  select *
  from employees as e
  where d.dpt = e.dpt and not(exists(
    select *
    from tasks as t
    where e.emp = t.emp and t.tsk = "abstract"))))
```

dpt
"Quality"
"Research"

# Importing the database

```
type Org = {departments : {dpt : string} list;  
            employees :  {dpt : string; emp : string} list;  
            tasks :      {emp : string; tsk : string} list }  
let org : Expr<Org> = <@ database("Org") @>
```

# Departments where every employee can do a given task

```
let expertise' : Expr< string → {dpt : string} list > =  
  <@ fun(u) → for d in (%org).departments do  
    if not(exists(  
      for e in (%org).employees do  
        if d.dpt = e.dpt && not(exists(  
          for t in (%org).tasks do  
            if e.emp = t.emp && t.tsk = u then yield { })  
        )) then yield { })  
    )) then yield {dpt = d.dpt} @>
```

```
run(<@ (%expertise')("abstract") @>  
[ {dpt = "Quality"}; {dpt = "Research"} ]
```

## Nested data

```
[ {dpt = "Product"; employees =  
  [ {emp = "Alex"; tasks = ["build"] }  
    {emp = "Bert"; tasks = ["build"] } ] ];  
{dpt = "Quality"; employees = [ ] };  
{dpt = "Research"; employees =  
  [ {emp = "Cora"; tasks = ["abstract"; "build"; "design"] };  
    {emp = "Drew"; tasks = ["abstract"; "design"] };  
    {emp = "Edna"; tasks = ["abstract"; "call"; "design"] } ] ];  
{dpt = "Sales"; employees =  
  [ {emp = "Fred"; tasks = ["call"] } ] ] ]
```





# Higher-order queries

**let any** : Expr < (A list, A → bool) → bool > =

<@ fun(xs, p) →

exists(for x in xs do

if p(x) then

yield { }) @>

**let all** : Expr < (A list, A → bool) → bool > =

<@ fun(xs, p) →

not((%any)(xs, fun(x) → not(p(x)))) @>

**let contains** : Expr < (A list, A) → bool > =

<@ fun(xs, u) →

(%any)(xs, fun(x) → x = u) @>

# Departments where every employee can do a given task

```
let expertise : Expr< string → {dpt : string} list > =  
  <@ fun(u) → for d in (%nestedOrg)  
    if (%all)(d.employees,  
      fun(e) → (%contains)(e.tasks, u) then  
    yield {dpt = d.dpt} @>
```

```
run(<@ (%expertise)("abstract") @>)
```

```
[ {dpt = "Quality"}; {dpt = "Research"} ]
```

# Normalisation: symbolic evaluation

**(fun**( $x$ )  $\rightarrow N$ )  $M \rightsquigarrow N[x := M]$

$\{\overline{\ell = M}\}.l_i \rightsquigarrow M_i$

**for**  $x$  **in** (yield  $M$ ) **do**  $N \rightsquigarrow N[x := M]$

**for**  $y$  **in** (for  $x$  **in**  $L$  **do**  $M$ ) **do**  $N \rightsquigarrow$  **for**  $x$  **in**  $L$  **do** (for  $y$  **in**  $M$  **do**  $N$ )

**for**  $x$  **in** (if  $L$  **then**  $M$ ) **do**  $N \rightsquigarrow$  **if**  $L$  **then** (for  $x$  **in**  $M$  **do**  $N$ )

**for**  $x$  **in** [ ] **do**  $N \rightsquigarrow$  [ ]

**for**  $x$  **in** ( $L$  @  $M$ ) **do**  $N \rightsquigarrow$  (for  $x$  **in**  $L$  **do**  $N$ ) @ (for  $x$  **in**  $M$  **do**  $N$ )

**if** true **then**  $M \rightsquigarrow M$

**if** false **then**  $M \rightsquigarrow$  [ ]

## Normalisation: *ad hoc* rewriting

**for**  $x$  **in**  $L$  **do**  $(M @ N)$   $\hookrightarrow$  (**for**  $x$  **in**  $L$  **do**  $M$ ) @ (**for**  $x$  **in**  $L$  **do**  $N$ )

**for**  $x$  **in**  $L$  **do**  $[\ ]$   $\hookrightarrow$   $[\ ]$

**if**  $L$  **then**  $(M @ N)$   $\hookrightarrow$  (**if**  $L$  **then**  $M$ ) @ (**if**  $L$  **then**  $N$ )

**if**  $L$  **then**  $[\ ]$   $\hookrightarrow$   $[\ ]$

**if**  $L$  **then** (**for**  $x$  **in**  $M$  **do**  $N$ )  $\hookrightarrow$  **for**  $x$  **in**  $M$  **do** (**if**  $L$  **then**  $N$ )

**if**  $L$  **then** (**if**  $M$  **then**  $N$ )  $\hookrightarrow$  **if** ( $L \ \&\& \ M$ ) **then**  $N$

**yield**  $x$   $\hookrightarrow$  **yield**  $\{\overline{\ell = x.\ell}\}$

**database**( $db$ ). $\ell$   $\hookrightarrow$  **for**  $x$  **in** **database**( $db$ ). $\ell$  **do** **yield**  $x$

## SQL LINQ results (F#)

Example	F# 2.0	F# 3.0	us	(norm)
differences	17.6	20.6	18.1	0.5
range	×	5.6	2.9	0.3
satisfies	2.6	×	2.9	0.3
P(t <sub>0</sub> )	2.8	×	3.3	0.3
P(t <sub>1</sub> )	2.7	×	3.0	0.3
expertise'	7.2	9.2	8.0	0.6
expertise	×	66.7 <sup>av</sup>	8.3	0.9
xp <sub>0</sub>	×	8.3	7.9	1.9
xp <sub>1</sub>	×	14.7	13.4	1.1
xp <sub>2</sub>	×	17.9	20.7	2.2
xp <sub>3</sub>	×	3744.9	3768.6	4.4

Times in milliseconds; <sup>av</sup> marks query avalanche.

The script-writers dream, Cooper, DBPL, 2009.

A practical theory of language integrated query,  
Cheney, Lindley, Wadler, ICFP, 2013.

Everything old is new again: Quoted Domain Specific Languages,  
Najd, Lindley, Svenningsson, Wadler, PEPM, 2016.

Propositions as types, Wadler, CACM, Dec 2015.

<http://fsprojects.github.io/FSharp.Linq.Experimental.ComposableQuery/>



Ezra Cooper<sup>\*†</sup>, James Cheney<sup>\*</sup>, Sam Lindley<sup>\*</sup>,  
Shayan Najd<sup>\*‡</sup>, Josef Svenningsson<sup>§</sup>, Philip Wadler<sup>\*</sup>

<sup>\*</sup>University of Edinburgh, <sup>†</sup>Qumulo, <sup>‡</sup>Google, <sup>§</sup>Chalmers & HiQ