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.

r

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

$$\begin{array}{lll} \textbf{type}\ \mathsf{Org} = \{ \mathsf{departments} : & \{\mathsf{dpt} : \textbf{string}\}\ \textbf{list}; \\ \quad \mathsf{employees} : & \{\mathsf{dpt} : \textbf{string}; \mathsf{emp} : \textbf{string}\}\ \textbf{list}; \\ \quad \mathsf{tasks} : & \{\mathsf{emp} : \textbf{string}; \mathsf{tsk} : \textbf{string}\}\ \textbf{list}\ \} \end{array}$$

$$\textbf{let}\ \mathsf{org} : \mathsf{Expr}\!<\mathsf{Org}> = \texttt{<@}\ \textbf{database}(\text{“Org”})\ \texttt{@>}$$

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

$$
\begin{array}{l}
[\{\mathsf{dpt} = \text{“Product”};\ \mathsf{employees} = \\
\quad [\{\mathsf{emp} = \text{“Alex”};\ \mathsf{tasks} = [\text{“build”}]\} \\
\quad\ \ \{\mathsf{emp} = \text{“Bert”};\ \mathsf{tasks} = [\text{“build”}]\}]\}; \\
\ \{\mathsf{dpt} = \text{“Quality”}; \mathsf{employees} = [\,]\}; \\
\ \{\mathsf{dpt} = \text{“Research”};\ \mathsf{employees} = \\
\quad [\{\mathsf{emp} = \text{“Cora”};\ \mathsf{tasks} = [\text{“abstract”}; \text{“build”}; \text{“design”}]\}; \\
\quad\ \ \{\mathsf{emp} = \text{“Drew”};\ \mathsf{tasks} = [\text{“abstract”}; \text{“design”}]\}; \\
\quad\ \ \{\mathsf{emp} = \text{“Edna”};\ \mathsf{tasks} = [\text{“abstract”}; \text{“call”}; \text{“design”}]\}]\}; \\
\ \{\mathsf{dpt} = \text{“Sales”};\ \mathsf{employees} = \\
\quad [\{\mathsf{emp} = \text{“Fred”};\ \mathsf{tasks} = [\text{“call”}]\}]\}]
\end{array}
$$

Nested data from flat data

```
type NestedOrg = [{dpt : string; employees :
                      [{emp : string; tasks : [string]}]}]
let nestedOrg : Expr<NestedOrg> =
  <@ for d in (%org).departments do
     yield {dpt = d.dpt; employees =
              for e in (%org).employees do
              if d.dpt = e.dpt then
              yield {emp = e.emp; tasks =
                       for t in (%org).tasks do
                       if e.emp = t.emp then
                       yield t.tsk}}} @>
```

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

$$
\begin{array}{rcl}
(\textbf{fun}(x) \to N)\ M & \rightsquigarrow & N[x := M] \\
\{\overline{\ell = M}\}.\ell_i & \rightsquigarrow & M_i \\
\textbf{for}\ x\ \textbf{in}\ (\textbf{yield}\ M)\ \textbf{do}\ N & \rightsquigarrow & N[x := M] \\
\textbf{for}\ y\ \textbf{in}\ (\textbf{for}\ x\ \textbf{in}\ L\ \textbf{do}\ M)\ \textbf{do}\ N & \rightsquigarrow & \textbf{for}\ x\ \textbf{in}\ L\ \textbf{do}\ (\textbf{for}\ y\ \textbf{in}\ M\ \textbf{do}\ N) \\
\textbf{for}\ x\ \textbf{in}\ (\textbf{if}\ L\ \textbf{then}\ M)\ \textbf{do}\ N & \rightsquigarrow & \textbf{if}\ L\ \textbf{then}\ (\textbf{for}\ x\ \textbf{in}\ M\ \textbf{do}\ N) \\
\textbf{for}\ x\ \textbf{in}\ \texttt{[ ]}\ \textbf{do}\ N & \rightsquigarrow & \texttt{[ ]} \\
\textbf{for}\ x\ \textbf{in}\ (L\ \texttt{@}\ M)\ \textbf{do}\ N & \rightsquigarrow & (\textbf{for}\ x\ \textbf{in}\ L\ \textbf{do}\ N)\ \texttt{@}\ (\textbf{for}\ x\ \textbf{in}\ M\ \textbf{do}\ N) \\
\textbf{if true then}\ M & \rightsquigarrow & M \\
\textbf{if false then}\ M & \rightsquigarrow & \texttt{[ ]}
\end{array}
$$

Normalisation: *ad hoc* rewriting

$$
\begin{array}{rcl}
\textbf{for}\ x\ \textbf{in}\ L\ \textbf{do}\ (M \mathbin{\texttt{@}} N) & \hookrightarrow & (\textbf{for}\ x\ \textbf{in}\ L\ \textbf{do}\ M) \mathbin{\texttt{@}} (\textbf{for}\ x\ \textbf{in}\ L\ \textbf{do}\ N) \\
\textbf{for}\ x\ \textbf{in}\ L\ \textbf{do}\ \texttt{[ ]} & \hookrightarrow & \texttt{[ ]} \\
\textbf{if}\ L\ \textbf{then}\ (M \mathbin{\texttt{@}} N) & \hookrightarrow & (\textbf{if}\ L\ \textbf{then}\ M) \mathbin{\texttt{@}} (\textbf{if}\ L\ \textbf{then}\ N) \\
\textbf{if}\ L\ \textbf{then}\ \texttt{[ ]} & \hookrightarrow & \texttt{[ ]} \\
\textbf{if}\ L\ \textbf{then}\ (\textbf{for}\ x\ \textbf{in}\ M\ \textbf{do}\ N) & \hookrightarrow & \textbf{for}\ x\ \textbf{in}\ M\ \textbf{do}\ (\textbf{if}\ L\ \textbf{then}\ N) \\
\textbf{if}\ L\ \textbf{then}\ (\textbf{if}\ M\ \textbf{then}\ N) & \hookrightarrow & \textbf{if}\ (L\ \ \&\&\ \ M)\ \textbf{then}\ N \\
\textbf{yield}\ x & \hookrightarrow & \textbf{yield}\ \{\overline{\ell = x.\ell}\} \\
\textbf{database}(\mathsf{db}).\ell & \hookrightarrow & \textbf{for}\ x\ \textbf{in}\ \textbf{database}(\mathsf{db}).\ell\ \textbf{do}\ \textbf{yield}\ x
\end{array}
$$

SQL LINQ results (F#)

Example	F# 2.0	F# 3.0	us	(norm)
differences	17.6	20.6	18.1	0.5
range	$\times$	5.6	2.9	0.3
satisfies	2.6	$\times$	2.9	0.3
$P(t_0)$	2.8	$\times$	3.3	0.3
$P(t_1)$	2.7	$\times$	3.0	0.3
expertise′	7.2	9.2	8.0	0.6
expertise	$\times$	66.7^{av}	8.3	0.9
xp_0	$\times$	8.3	7.9	1.9
xp_1	$\times$	14.7	13.4	1.1
xp_2	$\times$	17.9	20.7	2.2
xp_3	$\times$	3744.9	3768.6	4.4

Times in milliseconds; ^{av} 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[*,†], James Cheney[*], Sam Lindley[*],
Shayan Najd[*,‡], Josef Svenningsson[§], Philip Wadler[*]
[*]University of Edinburgh, [†]Qumulo, [‡]Google, [§]Chalmers & HiQ