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.”

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

<table>
<thead>
<tr>
<th>departments</th>
<th>employees</th>
<th>tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>dpt</td>
<td>dpt</td>
<td>emp</td>
</tr>
<tr>
<td>“Product”</td>
<td>“Product”</td>
<td>“Alex”</td>
</tr>
<tr>
<td>“Quality”</td>
<td>“Product”</td>
<td>“Bert”</td>
</tr>
<tr>
<td>“Research”</td>
<td>“Research”</td>
<td>“Cora”</td>
</tr>
<tr>
<td>“Sales”</td>
<td>“Research”</td>
<td>“Drew”</td>
</tr>
<tr>
<td></td>
<td>“Sales”</td>
<td>“Edna”</td>
</tr>
<tr>
<td></td>
<td></td>
<td>“Fred”</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

|             | 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

```sql
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"))))
```

<table>
<thead>
<tr>
<th>dpt</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Quality”</td>
</tr>
<tr>
<td>“Research”</td>
</tr>
</tbody>
</table>
Importing the database

```plaintext
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

```ocaml
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

```javascript
[ {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”] } ] ] }
```
Nested data from flat data

```r
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

```plaintext
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

\[
(fun(x) \rightarrow N) \; M \; \leadsto \; N[x := M] \\
\{ \ell = M \}.\ell_i \; \leadsto \; M_i \\
\text{for } x \text{ in (yield } M \text{) do } N \; \leadsto \; N[x := M] \\
\text{for } y \text{ in (for } x \text{ in } L \text{ do } M \text{) do } N \; \leadsto \; \text{for } x \text{ in } L \text{ do (for } y \text{ in } M \text{ do } N) \\
\text{for } x \text{ in (if } L \text{ then } M \text{) do } N \; \leadsto \; \text{if } L \text{ then (for } x \text{ in } M \text{ do } N) \\
\text{for } x \text{ in } [ ] \text{ do } N \; \leadsto \; [ ] \\
\text{for } x \text{ in } (L @ M) \text{ do } N \; \leadsto \; (\text{for } x \text{ in } L \text{ do } N) @ (\text{for } x \text{ in } M \text{ do } N) \\
\text{if true then } M \; \leadsto \; M \\
\text{if false then } M \; \leadsto \; [ ]
\]
Normalisation: *ad hoc* rewriting

\[
\text{for } x \text{ in } L \text{ do } (M @ N) \leftrightarrow (\text{for } x \text{ in } L \text{ do } M) @ (\text{for } x \text{ in } L \text{ do } N)
\]

\[
\text{for } x \text{ in } L \text{ do } [ ] \leftrightarrow [ ]
\]

\[
\text{if } L \text{ then } (M @ N) \leftrightarrow (\text{if } L \text{ then } M) @ (\text{if } L \text{ then } N)
\]

\[
\text{if } L \text{ then } [ ] \leftrightarrow [ ]
\]

\[
\text{if } L \text{ then } (\text{for } x \text{ in } M \text{ do } N) \leftrightarrow \text{for } x \text{ in } M \text{ do } (\text{if } L \text{ then } N)
\]

\[
\text{if } L \text{ then } (\text{if } M \text{ then } N) \leftrightarrow \text{if } (L \&\& M) \text{ then } N
\]

\[
\text{yield } x \leftrightarrow \text{yield } \{ \ell = x.\ell \}
\]

\[
\text{database(db).l} \leftrightarrow \text{for } x \text{ in database(db).l \ do \ yield } x
\]
### SQL LINQ results (F#)

<table>
<thead>
<tr>
<th>Example</th>
<th>F# 2.0</th>
<th>F# 3.0</th>
<th>us</th>
<th>(norm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>differences</td>
<td>17.6</td>
<td>20.6</td>
<td>18.1</td>
<td>0.5</td>
</tr>
<tr>
<td>range</td>
<td>×</td>
<td>5.6</td>
<td>2.9</td>
<td>0.3</td>
</tr>
<tr>
<td>satisfies</td>
<td>2.6</td>
<td>×</td>
<td>2.9</td>
<td>0.3</td>
</tr>
<tr>
<td>P(t₀)</td>
<td>2.8</td>
<td>×</td>
<td>3.3</td>
<td>0.3</td>
</tr>
<tr>
<td>P(t₁)</td>
<td>2.7</td>
<td>×</td>
<td>3.0</td>
<td>0.3</td>
</tr>
<tr>
<td>expertise'</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>expertise</td>
<td>×</td>
<td>66.7av</td>
<td>8.3</td>
<td>0.9</td>
</tr>
<tr>
<td>xp₀</td>
<td>×</td>
<td>8.3</td>
<td>7.9</td>
<td>1.9</td>
</tr>
<tr>
<td>xp₁</td>
<td>×</td>
<td>14.7</td>
<td>13.4</td>
<td>1.1</td>
</tr>
<tr>
<td>xp₂</td>
<td>×</td>
<td>17.9</td>
<td>20.7</td>
<td>2.2</td>
</tr>
<tr>
<td>xp₃</td>
<td>×</td>
<td>3744.9</td>
<td>3768.6</td>
<td>4.4</td>
</tr>
</tbody>
</table>

Times in milliseconds; `av` marks query avalanche.
The script-writers dream, Cooper, DBPL, 2009.


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

Propositions as types, Wadler, CACM, Dec 2015.


Ezra Cooper∗†, James Cheney*, Sam Lindley*, Shayan Najd∗‡, Josef Svenningsson§, Philip Wadler*

*University of Edinburgh, †Qumulo, ‡Google, §Chalmers & HiQ