

Everything old is new again:
Quoted Domain Specific Languages

Philip Wadler
University of Edinburgh

DSLDI

Prague, Tuesday 7 July 2015

How does one integrate a Domain-Specific Language
and a host language?

Quotation (McCarthy, 1960)

Normalisation (Gentzen, 1935)

Part I

Getting started: Join queries

A query: Who is younger than Alex?

people

name	age
"Alex"	40
"Bert"	30
"Cora"	35
"Drew"	60
"Edna"	25
"Fred"	70

```
select v.name as name,  
         v.age as age  
from people as u,  
      people as v  
where u.name = "Alex" and  
       v.age < u.age
```

answer

name	age
"Bert"	30
"Cora"	35
"Edna"	25

A database as data

people

name	age
"Alex"	40
"Bert"	30
"Cora"	35
"Drew"	60
"Edna"	25
"Fred"	70

{people =

```
[ {name = "Alex" ; age = 40};  
  {name = "Bert" ; age = 30};  
  {name = "Cora" ; age = 35};  
  {name = "Drew"; age = 60};  
  {name = "Edna"; age = 25};  
  {name = "Fred" ; age = 70} ] }
```

A query as F# code (naive)

```
type DB = {people : {name : string; age : int} list}  
let db' : DB = database("People")  
let youths' : {name : string; age : int} list =  
    for u in db'.people do  
        for v in db'.people do  
            if u.name = "Alex" && v.age < u.age then  
                yield {name : v.name; age : v.age}
```

youths' \rightsquigarrow

```
[ {name = "Bert" ; age = 30}  
  {name = "Cora" ; age = 35}  
  {name = "Edna"; age = 25} ]
```

A query as F# code (quoted)

```
type DB = {people : {name : string; age : int} list}  
let db : Expr< DB > = <@ database("People") @>  
let youths : Expr< {name : string; age : int} list > =  
  <@ for u in (%db).people do  
    for v in (%db).people do  
      if u.name = "Alex" && v.age < u.age then  
        yield {name : v.name; age : v.age} @>
```

run(youths) \rightsquigarrow

```
[ {name = "Bert" ; age = 30}  
  {name = "Cora"; age = 35}  
  {name = "Edna"; age = 25} ]
```

What does **run** do?

1. Simplify quoted expression
2. Translate query to SQL
3. Execute SQL
4. Translate answer to host language

Theorem

Each **run** generates one query if

- A. answer type is flat (list of record of scalars)
- B. only permitted operations (e.g., no recursion)
- C. only refers to one database

Scala (naive)

```
val youth' : List [ { val name : String; val age : Int } ] =  
  for { u ← db'.people  
    v ← db'.people  
    if u.name == "Alex" && v.age < u.age }  
  yield new Record { val name = v.name; val age = v.age }
```

Scala (quoted)

```
val youth : Rep [ List [ { val name : String; val age : Int } ] ] =  
  for { u ← db.people  
    v ← db.people  
    if u.name == "Alex" && v.age < u.age }  
  yield new Record { val name = v.name; val age = v.age }
```

Part II

Nested intermediate data

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"

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"] } ] ] ]
```

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"} ]
```

Part III

Conclusion

How does one integrate a Domain-Specific Language
and a host language?

Quotation (McCarthy, 1960)

Normalisation (Gentzen, 1935)

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, Draft, 2015.

Propositions as types, Wadler, CACM, to appear.

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



Ezra Cooper^{*†}, James Cheney^{*}, Sam Lindley^{*},
Shayan Najd^{*†}, Josef Svenningsson[‡], Philip Wadler^{*}

^{*}University of Edinburgh, [†] Google, [‡]Chalmers University