<Programming> 2018
Nice, France

Language-integrated Provenance in Haskell

Jan Stolarek James Cheney
University of Edinburgh

Provenance

Tracing the origin of data.

This talk will focus on provenance in database context.

agencies

id	name	based_in	phone
1	EdinTours	Edinburgh	412 1200
2	Burns's	Glasgow	607 3000

externaltours

id	name	destination	type
3	EdinTours	Edinburgh	bus
4	EdinTours	Loch Ness	bus
5	EdinTours	Loch Ness	boat
6	EdinTours	Firth of Forth	boat
7	Burns's	Islay	boat
8	Burns's	Mallaig	train

Query: names and phone numbers of agencies organizing boat tours

agencies

id	name	based_in	phone
1	EdinTours	Edinburgh	412 1200
2	Burns's	Glasgow	607 3000

externaltours

id	name	destination	type
3	EdinTours	Edinburgh	bus
4	EdinTours	Loch Ness	bus
5	EdinTours	Loch Ness	boat
6	EdinTours	Firth of Forth	boat
7	Burns's	Islay	boat
8	Burns's	Mallaig	train

```
SELECT et.name, a.phone
FROM agencies AS a, externaltours AS et
WHERE (a.name = et.name) AND (et.type = 'boat')
```

agencies

id	name	based_in	phone
1	EdinTours	Edinburgh	412 1200
2	Burns's	Glasgow	607 3000

externaltours

id	name	destination	type
3	EdinTours	Edinburgh	bus
4	EdinTours	Loch Ness	bus
5	EdinTours	Loch Ness	boat
6	EdinTours	Firth of Forth	boat
7	Burns's	Islay	boat
8	Burns's	Mallaig	train

result

id	name	phone
1	EdinTours	412 1200
2	EdinTours	412 1200
3	Burns's	607 3000

agencies

id	name	based_in	phone
1	EdinTours	Edinburgh	412 1200
2	Burns's	Glasgow	607 3000

externaltours

id	name	destination	type
3	EdinTours	Edinburgh	bus
4	EdinTours	Loch Ness	bus
5	EdinTours	Loch Ness	boat
6	EdinTours	Firth of Forth	boat
7	Burns's	Islay	boat
8	Burns's	Mallaig	train

result

id	name	phone with where-provenance tracking
1	EdinTours	(data = 412 1200, prov = ("agencies", "phone", 1))
2	EdinTours	(data = 412 1200, prov = ("agencies", "phone", 1))
3	Burns's	(data = 607 3000, prov = ("agencies", "phone", 2))

agencies

id	name	based_in	phone
1	EdinTours	Edinburgh	412 1200
2	Burns's	Glasgow	607 3000

externaltours

id	name	destination	type
3	EdinTours	Edinburgh	bus
4	EdinTours	Loch Ness	bus
5	EdinTours	Loch Ness	boat
6	EdinTours	Firth of Forth	boat
7	Burns's	Islay	boat
8	Burns's	Mallaig	train

result

id	name	phone with where-provenance tracking
1	EdinTours	(data = 412 1200, prov = ("agencies", "phone", 1))
2	EdinTours	(data = 412 1200, prov = ("agencies", "phone", 1))
3	Burns's	(data = 607 3000, prov = ("agencies", "phone", 2))

agencies

id	name	based_in	phone
1	EdinTours	Edinburgh	412 1200
2	Burns's	Glasgow	607 3000

externaltours

id	name	destination	type
3	EdinTours	Edinburgh	bus
4	EdinTours	Loch Ness	bus
5	EdinTours	Loch Ness	boat
6	EdinTours	Firth of Forth	boat
7	Burns's	Islay	boat
8	Burns's	Mallaig	train

result

id	name	phone	lineage
1	EdinTours	412 1200	[("agencies", 1), ("externaltours", 5)]
2	EdinTours	412 1200	[("agencies", 1), ("externaltours", 6)]
3	Burns's	607 3000	[("agencies", 2), ("externaltours", 7)]

agencies

id	name	based_in	phone
1	EdinTours	Edinburgh	412 1200
2	Burns's	Glasgow	607 3000

externaltours

id	name	destination	type
3	EdinTours	Edinburgh	bus
4	EdinTours	Loch Ness	bus
5	EdinTours	Loch Ness	boat
6	EdinTours	Firth of Forth	boat
7	Burns's	Islay	boat
8	Burns's	Mallaig	train

result

id	name	phone	lineage
1	EdinTours	412 1200	[("agencies", 1), ("externaltours", 5)]
2	EdinTours	412 1200	[("agencies", 1), ("externaltours", 6)]
3	Burns's	607 3000	[("agencies", 2), ("externaltours", 7)]

agencies

id	name	based_in	phone
1	EdinTours	Edinburgh	412 1200
2	Burns's	Glasgow	607 3000

externaltours

id	name	destination	type
3	EdinTours	Edinburgh	bus
4	EdinTours	Loch Ness	bus
5	EdinTours	Loch Ness	boat
6	EdinTours	Firth of Forth	boat
7	Burns's	Islay	boat
8	Burns's	Mallaig	train

result

id	name	phone	lineage
1	EdinTours	412 1200	[("agencies", 1), ("externaltours", 5)]
2	EdinTours	412 1200	[("agencies", 1), ("externaltours", 6)]
3	Burns's	607 3000	[("agencies", 2), ("externaltours", 7)]

Provenance

Still an experimental feature found only in research prototypes (e.g. Links).

Our goal: provenance as library.

Our approach: extend Database Supported Haskell (DSH) library with provenance support.

Haskell

In this talk Haskell = GHC (Glasgow Haskell Compiler)

Haskell is a purely functional, statically typed programming language.

It has a rich and expressive type system.

Good for implementing Embedded Domain-Specific Languages (EDSLs).

DSH created by Torsten Grust and Alexander Ulrich[1].

Provides language-integrated queries in Haskell by overloading list comprehension notation.

[1] "The Flatter, the Better: Query Compilation Based on the Flattening Transformation", ACM SIGMOD 2015

agencies

id	name	based_in	phone
1	EdinTours	Edinburgh	412 1200
2	Burns's	Glasgow	607 3000

```
data Agency = Agency { a_id       :: Integer
                     , a_name     :: String
                     , a_based_in :: String
                     , a_phone    :: String }
```

```
agencies :: Q [Agency]
agencies = table "agencies"
              [ "id", "name", "based_in", "phone" ]
             (TableHints [Key [ "id" ]])
```

agencies

id	name	based_in	phone
1	EdinTours	Edinburgh	412 1200
2	Burns's	Glasgow	607 3000

```
data Agency = Agency { a_id        :: Integer
                     , a_name      :: String
                     , a_based_in  :: String
                     , a_phone     :: String }
```

```
agencies :: Q [Agency]
agencies = table "agencies"
               [ "id", "name", "based_in", "phone" ]
              (TableHints [Key [ "id" ]])
```

agencies

id	name	based_in	phone
1	EdinTours	Edinburgh	412 1200
2	Burns's	Glasgow	607 3000

```
data Agency = Agency { a_id        :: Integer
                     , a_name      :: String
                     , a_based_in  :: String
                     , a_phone     :: String }
```

```
agencies :: Q [Agency]
agencies = table "agencies"
             [ "id", "name", "based_in", "phone" ]
             (TableHints [Key [ "id" ]])
```

agencies

id	name	based_in	phone
1	EdinTours	Edinburgh	412 1200
2	Burns's	Glasgow	607 3000

```
data Agency = Agency { a_id       :: Integer
                     , a_name     :: String
                     , a_based_in :: String
                     , a_phone    :: String }
```

```
agencies :: Q [Agency]
agencies = table  "agencies"
           [ "id", "name", "based_in", "phone" ]
           (TableHints [Key [ "id" ]])
```

agencies

id	name	based_in	phone
1	EdinTours	Edinburgh	412 1200
2	Burns's	Glasgow	607 3000

```
data Agency = Agency { a_id        :: Integer
                     , a_name      :: String
                     , a_based_in  :: String
                     , a_phone     :: String }
```

```
agencies :: Q [Agency]
agencies = table "agencies"
               [ "id", "name", "based_in", "phone" ]
               (TableHints [Key [ "id" ]])
```

Database Supported Haskell example

```
q1 :: Q [(String, String)]
q1 = [ (et_name et, a_phone a)
     | a  <- agencies
     , et <- externaltours
     , a_name a == et_name et
     , et_type et == "boat" ]
```

```
SELECT a1.name AS i1, a0.phone AS i2
FROM agencies AS a0, externaltours AS a1
WHERE (a0.name = a1.name) AND (a1.type = 'boat')
```

Lineage transformation

Lineage tracing obtained by calling a library function on an existing query:

```
q1L :: Q (LT [(String, String)] Integer)
q1L = lineage q1
```

DSH compilation pipeline

1. **Haskell source.** GHC extensions + DSH module imports
2. **Frontend Language (FL).** Typing invariants embedded inside Haskell's type system.
3. **Additional translations and compilation to SQL.**

Lineage transformation:

- follows query rewriting approach developed in Links
- global rewriting of the whole syntax tree
- accompanied by a type translation

```
q1L :: Q (LT [(String, String)] Integer)
q1L = lineage q1
```

$$
\begin{aligned}
\mathfrak{L}_\theta(\mathbf{table}_{(n:\mathsf{String},\phi:R\to\theta)}) &= \text{[redacted]}\\
\mathfrak{L}_\theta(\mathbf{concatMap}\ (\lambda f.M)\ xs) &= \text{[redacted]}\\
\mathfrak{L}_\theta(\mathbf{map}\ (\lambda f.M)\ xs) &= \text{[redacted]}\\
\mathfrak{L}_\theta(\mathbf{append}\ xs\ ys) &= \text{[redacted]}\\
\mathfrak{L}_\theta(\mathbf{reverse}\ xs) &= \text{[redacted]}\\
\mathfrak{L}_\theta(\mathbf{zip}\ xs\ ys) &= \text{[redacted]}\\
\mathfrak{L}_\theta(c) &= \text{[redacted]}\\
\mathfrak{L}_\theta(x) &= \text{[redacted]}\\
\mathfrak{L}_\theta([M_1,\ldots,M_n]) &= \text{[redacted]}\\
\mathfrak{L}_\theta(\mathbf{cons}\ x\ xs) &= \text{[redacted]}\\
\mathfrak{L}_\theta(\mathbf{guard}\ b) &= \text{[redacted]}\\
\mathfrak{L}_\theta(M_1,\ldots,M_n) &= \text{[redacted]}\\
\mathfrak{L}_\theta(M.n) &= \text{[redacted]}
\end{aligned}
$$

$$
\begin{array}{rcl}
\mathfrak{L}([\delta]) & = & [\mathfrak{L}(\delta)^{L}] \\
\mathfrak{L}(\delta_1, \ldots, \delta_n) & = & (\mathfrak{L}(\delta_1), \ldots, \mathfrak{L}(\delta_n)) \\
\mathfrak{L}(\mathbf{()}) & = & \mathbf{()} \\
\mathfrak{L}(\textbf{String}) & = & \textbf{String} \\
\mathfrak{L}(\textbf{Bool}) & = & \textbf{Bool} \\
\mathfrak{L}(\textbf{Int}) & = & \textbf{Int}
\end{array}
$$

Summary

Our achievements:

- implementation of provenance tracking as part of a library, rather than as part of a compiler
- maintaining all the benefits of DSH type safety

Summary

Open questions and further research:

- challenging to express typing rules in the EDSL approach; perhaps a different meta-programming technique would be better?
- how to replicate this result in other languages (Scala, F#)?

Summary

More in the paper:

- where-provenance tracking
- details on surface encodings of provenance
- formal specification of FL, where-provenance and lineage transformations
- detailed description of DSH implementation and explanation of technical challenges

Implementation available at:

`https://github.com/jstolarek/skye-dsh`

<Programming> 2018

Nice, France

Language-integrated Provenance in Haskell

Jan Stolarek James Cheney

University of Edinburgh

$$
\begin{array}{rcl}
\mathfrak{L}_\theta(\mathbf{table}_{(n:\mathbf{String},\phi:R\to\theta)}) & = & \mathbf{map}\ (\lambda x.x^{(t,\phi(x))})\ \mathbf{table}_{(t:\mathbf{String},\phi:R\to\theta)} \\
\mathfrak{L}_\theta(\mathbf{concatMap}\ (\lambda f.M)\ xs) & = & \mathbf{concatMap}\ (\lambda x. \\
& & \quad \mathbf{map}\ (\lambda z.(z.\mathbf{data}^{z.\mathbf{prov}\oplus x.\mathbf{prov}}))\ \mathfrak{L}_\theta((\lambda f.M)(x.\mathbf{data}))) \\
& & \quad \mathfrak{L}_\theta(xs) \\
\mathfrak{L}_\theta(\mathbf{map}\ (\lambda f.M)\ xs) & = & \mathbf{concatMap}\ (\lambda x. \\
& & \quad \mathbf{map}\ (\lambda z.(z.\mathbf{data}^{z.\mathbf{prov}\oplus x.\mathbf{prov}}))\ \mathfrak{L}_\theta[(\lambda f.M)(x.\mathbf{data})]) \\
& & \quad \mathfrak{L}_\theta(xs) \\
\mathfrak{L}_\theta(\mathbf{append}\ xs\ ys) & = & \mathbf{append}\ (\mathfrak{L}_\theta(xs))\ (\mathfrak{L}_\theta(ys)) \\
\mathfrak{L}_\theta(\mathbf{reverse}\ xs) & = & \mathbf{reverse}\ (\mathfrak{L}_\theta(xs)) \\
\mathfrak{L}_\theta(\mathbf{zip}\ xs\ ys) & = & \mathbf{map}\ (\lambda x.(x.1.\mathbf{data}, x.2.\mathbf{data})^{x.1.\mathbf{prov}\oplus x.2.\mathbf{prov}}) \\
& & \quad (\mathbf{zip}\ (\mathfrak{L}_\theta(xs))\ (\mathfrak{L}_\theta(ys))) \\
\mathfrak{L}_\theta(c) & = & c \\
\mathfrak{L}_\theta(x) & = & x \\
\mathfrak{L}_\theta([M_1,\ldots,M_n]) & = & \mathbf{map}\ (\lambda x.x^\perp)\ [\mathfrak{L}_\theta(M_1),\ldots,\mathfrak{L}_\theta(M_n)] \\
\mathfrak{L}_\theta(\mathbf{cons}\ x\ xs) & = & \mathbf{cons}\ (\mathfrak{L}_\theta(x)^\perp)\ (\mathfrak{L}_\theta(xs)) \\
\mathfrak{L}_\theta(\mathbf{guard}\ b) & = & \mathbf{map}\ (\lambda x.x^\perp)\ (\mathbf{guard}\ b) \\
\mathfrak{L}_\theta(M_1,\ldots,M_n) & = & (\mathfrak{L}_\theta(M_1),\ldots,\mathfrak{L}_\theta(M_n)) \\
\mathfrak{L}_\theta(M.n) & = & (\mathfrak{L}_\theta(M)).n
\end{array}
$$