A fun interpreter of the $\mathcal{RL}$-language

Romuald THION

June 17, 2014

Abstract

This document is an implementation of the language defined in [AFP+11] written in the literate programming style [Knu84]. Thus, it can be seen as both:

- a mathematical definition of a formal language, with denotational semantics,
- a proof of concept interpreter.

We use the Haskell programming language: a pure, non-strict, lazy, funtional language [PJHA+99]. The document is meant to compile without warning with full strictures turned on.

Contents

1 Syntax

1.1 Basic ingredients

We give basic ingredients of $\mathcal{RL}$-language. The set of attributes $\mathcal{U}$ is assumed to be enumerable. A schema R is a subset of $\mathcal{U}$. The set of schemas is implemented as a type *Sch a* over an enumerable domain (*Enum a* constraints).

type $\mathcal{CST} = \mathbb{N}$
-- the set of all constants

data *Sch a* **where**
$\mathit{Sch} :: (\mathit{Enum}\ a) \Rightarrow a \rightarrow a \rightarrow \mathit{Sch}\ a$
-- a schema is a finite subset of attributes
-- parameters are the minimum and maximum bound
$:: \mathit{Sch}\ a \rightarrow [a]$
$(\mathit{Sch}\ a\ b) = \mathit{enumFromTo}\ a\ b$

data *Tuple a* **where**
$\mathit{Tuple} :: \mathit{Sch}\ a \rightarrow (a \rightarrow \mathcal{CST}) \rightarrow \mathit{Tuple}\ a$
-- a tuple is a total function from a given schema into constants

$\mathit{mkTuple} :: (\mathit{Eq}\ a) \Rightarrow \mathit{Sch}\ a \rightarrow [\mathcal{CST}] \rightarrow \mathit{Tuple}\ a$
$\mathit{mkTuple}\ s = \mathbf{let}\ f = (\lambda x \rightarrow \mathit{maybe}\ \bot\ \mathit{id} \circ x) \circ (\mathit{flip}\ \mathit{lookup}) \circ \mathit{zip}\ (\ s)\ \mathbf{in}\ (\mathit{Tuple}\ s) \circ f$
-- build a tuple from a given list of constants

$\cdot[\cdot] :: \mathit{Tuple}\ a \rightarrow a \rightarrow \mathcal{CST}$
$(\mathit{Tuple}\ \_\ f)[x] = f\ x$
-- projection function

type $\mathit{TVar} = \mathit{String}$
-- tuple-variables
type $\mathit{AVar} = \mathit{String}$
-- atttribute-variables
type $\mathit{SVar} = \mathit{String}$
-- schema-variables

1.2 Atomic & $\mathcal{RL}$ formulas

type $\mathit{Proj} = (\mathit{TVar}, \mathit{AVar})$
data $\mathcal{A}_a = (=)\ \mathit{AVar}\ \mathit{AVar}$
$\mid (=)\ \mathit{Proj}\ \mathit{Proj}$
$\mid (=)\ \mathit{Proj}\ \mathcal{CST}$
$\mid (=)\ \mathit{AVar}\ a$
$\mid (\in)\ \mathit{AVar}\ \mathit{SVar}$
deriving $(\mathit{Eq}, \mathit{Show})$

Formulas of the $\mathcal{RL}$-language, syntatic expressions will be desugared in the definition of the semantics:

data $\mathcal{RL}_a = (\mathcal{A}_a)$
$\mid (\wedge)\ (\mathcal{RL}_a)\ (\mathcal{RL}_a)$
$\mid (\vee)\ (\mathcal{RL}_a)\ (\mathcal{RL}_a)$
$\mid (\Rightarrow)\ (\mathcal{RL}_a)\ (\mathcal{RL}_a)$

$$
\begin{aligned}
&\mid \neg\ (\mathcal{RL}_a) \\
&\mid \forall AVar.\ (\mathcal{RL}_a) \\
&\mid \exists AVar.\ (\mathcal{RL}_a) \\
&\mid \forall TVar.\ (\mathcal{RL}_a) \\
&\mid \exists TVar.\ (\mathcal{RL}_a) \\
&\mid \forall AVar \in SVar.\ (\mathcal{RL}_a) \\
&\mid \exists AVar \in SVar.\ (\mathcal{RL}_a) \\
&\mid |TVar| \geqslant \mathbb{N}.\ (\mathcal{RL}_a)
\end{aligned}
$$

$\textbf{deriving}\ (Eq, Show)$

1.3 Extraction of variables

$\textbf{type}\ VarSet = (Set\ TVar, Set\ AVar, Set\ SVar)$

$(\uplus) :: VarSet \rightarrow VarSet \rightarrow VarSet$
$(a,b,c) \uplus (x,y,z) = (a \cup x, b \cup y, c \cup z)$

$$
\begin{aligned}
&FV^{\mathcal{A}}(\cdot) :: \mathcal{A}_a \rightarrow VarSet \\
&FV^{\mathcal{A}}(((u,a) = (v,b))) = (fromList\ [u,v], fromList\ [a,b], \emptyset) \\
&FV^{\mathcal{A}}(((u,a) = \_)) = (\{u\}, \{a\}, \emptyset) \\
&FV^{\mathcal{A}}((a = b)) = (\emptyset, fromList\ [a,b], \emptyset) \\
&FV^{\mathcal{A}}((a = \_)) = (\emptyset, \{a\}, \emptyset) \\
&FV^{\mathcal{A}}((a \in x)) = (\emptyset, \{a\}, \{x\})
\end{aligned}
$$

$$
\begin{aligned}
&FV(\cdot) :: \mathcal{RL}_a \rightarrow VarSet \\
&FV((a)) = FV^{\mathcal{A}}(a) \\
&FV((x \wedge y)) = FV(x) \uplus FV(y) \\
&FV((x \vee y)) = FV(x) \uplus FV(y) \\
&FV((x \Rightarrow y)) = FV(x) \uplus FV(y) \\
&FV((\neg\ x)) = FV(x) \\
&FV((\forall a.\ x)) = \textbf{let}\ (t', a', s') = FV(x)\ \textbf{in}\ (t', a' \setminus \{a\}, s') \\
&FV((\exists a.\ x)) = \textbf{let}\ (t', a', s') = FV(x)\ \textbf{in}\ (t', a' \setminus \{a\}, s') \\
&FV((\forall t.\ x)) = \textbf{let}\ (t', a', s') = FV(x)\ \textbf{in}\ (t' \setminus \{t\}, a', s') \\
&FV((\exists t.\ x)) = \textbf{let}\ (t', a', s') = FV(x)\ \textbf{in}\ (t' \setminus \{t\}, a', s') \\
&FV((\forall a \in s.\ x)) = \textbf{let}\ (t', a', s') = FV(x)\ \textbf{in}\ (t', a' \setminus \{a\}, s' \cup \{s\}) \\
&FV((\exists a \in s.\ x)) = \textbf{let}\ (t', a', s') = FV(x)\ \textbf{in}\ (t', a' \setminus \{a\}, s' \cup \{s\}) \\
&FV((|t| \geqslant \_.\ x)) = \textbf{let}\ (t', a', s') = FV(x)\ \textbf{in}\ (t' \setminus \{t\}, a', s')
\end{aligned}
$$

2 Semantics

$\textbf{type}\ RLStruct\ a = (Sch\ a, [Tuple\ a], (SVar \rightarrow [a]))$
$\textbf{type}\ RLInter\ a = (RLStruct\ a, ((TVar \rightarrow Tuple\ a), (AVar \rightarrow a)))$

2.1 Satisfaction of atomic formulas

We split the definition of satisfaction into:

- satisfaction for atomic formulas $[\![\cdot]\!]^{\mathcal{A}}_{\cdot}$

- satisfaction for recurive formulas $[\![\cdot]\!]$.

$$
\begin{array}{lll}
[\![\cdot]\!]^{\mathcal{A}}_{\cdot} :: (Eq\ a) \Rightarrow (RLInter\ a) \rightarrow (\mathcal{A}_a) \rightarrow \mathbb{B} & & \\
[\![((u,a) = (v,b))]\!]^{\mathcal{A}}_{((\_,\_,\_),(\sigma^1,\sigma^2))} & = (\sigma^1\ u)[(\sigma^2\ a)] & \Leftrightarrow (\sigma^1\ v)[(\sigma^2\ b)] \\
[\![((u,a) = c)]\!]^{\mathcal{A}}_{((\_,\_,\_),(\sigma^1,\sigma^2))} & = (\sigma^1\ u)[(\sigma^2\ a)] & \Leftrightarrow c \\
[\![(a = b)]\!]^{\mathcal{A}}_{((\_,\_,\_),(\_,\sigma^2))} & = (\sigma^2\ a) & \Leftrightarrow (\sigma^2\ b) \\
[\![(a = c)]\!]^{\mathcal{A}}_{((\_,\_,\_),(\_,\sigma^2))} & = (\sigma^2\ a) & \Leftrightarrow c \\
[\![(a \in x)]\!]^{\mathcal{A}}_{((\_,\_,\Sigma),(\_,\sigma^2))} & = (\sigma^2\ a) & \in\ (\Sigma\ x)
\end{array}
$$

- Note the absence of the schema in the orgininal definition: it is added here.
- Note that σ very looks like *an interpretation* of variables, splitted into two parts.
- Non-traditional part is the *knot* with the $X(A)$ construct. BTW, I do think that $A \in X$ will be better.

2.2 Satisfaction of $\mathcal{RL}$ formulas

$$
\begin{array}{ll}
[\![\cdot]\!]_{\cdot} :: (Eq\ a) \Rightarrow (RLInter\ a) \rightarrow (\mathcal{RL}_a) \rightarrow \mathbb{B} & \\
[\![(a)]\!]_{st} & = [\![a]\!]^{\mathcal{A}}_{st} \\
[\![(x \wedge y)]\!]_{st} & = ([\![x]\!]_{st}) \wedge ([\![y]\!]_{st}) \\
[\![(x \vee y)]\!]_{st} & = ([\![x]\!]_{st}) \vee ([\![y]\!]_{st}) \\
[\![(x \Rightarrow y)]\!]_{st} & = ([\![\varphi]\!]_{st})\ \textbf{where} \\
 & \quad \varphi = (\neg\ x) \vee y \\
[\![(\neg\ x)]\!]_{st} & = \neg\ ([\![x]\!]_{st})
\end{array}
$$

-- end of classical logic

-- note that it is not proven that material implication and De Morgan laws are true !

$[\![(\forall a.\ x)]\!]_{((R,r,\Sigma),(\sigma^1,\sigma^2))} = foldr\ (\wedge)\ \textbf{tt}\ bs\ \textbf{where}$

-- schema (set) is needed here !!

$fs = allAtts\ R\ \sigma^2\ a$

-- fs is the list of all functions that update σ^2

$ss = upAtts\ fs\ ((R,r,\Sigma),(\sigma^1,\bot))$

-- we build the structures

$bs = satAll\ ss\ x$

-- we compute the evalutation of x according to each structure in the list

$[\![(\exists a.\ x)]\!]_{((R,r,\Sigma),(\sigma^1,\sigma^2))} = foldr\ (\vee)\ \textbf{ff}\ bs\ \textbf{where}$

$fs = allAtts\ R\ \sigma^2\ a$

$ss = upAtts\ fs\ ((R,r,\Sigma),(\sigma^1,\bot))$

$bs = satAll\ ss\ x$

$[\![(\forall t.\ x)]\!]_{((R,r,\Sigma),(\sigma^1,\sigma^2))} = foldr\ (\wedge)\ \textbf{tt}\ bs\ \textbf{where}$

-- similar to previous case, with update of σ^1

$fs = allTups\ r\ \sigma^1\ t$

$ss = upTups\ fs\ ((R,r,\Sigma),(\bot,\sigma^2))$

$bs = satAll\ ss\ x$

$[\![(\exists t.\ x)]\!]_{((R,r,\Sigma),(\sigma^1,\sigma^2))} = foldr\ (\vee)\ \textbf{ff}\ bs\ \textbf{where}$

$fs = allTups\ r\ \sigma^1\ t$

$ss = upTups\ fs\ ((R,r,\Sigma),(\bot,\sigma^2))$

$$
\begin{aligned}
&\qquad\qquad\qquad bs = satAll\ ss\ x \\
&[\![(\forall a \in s.\ x)]\!]_{st} = ([\![\varphi]\!]_{st})\ \textbf{where} \\
&\qquad\qquad \varphi = \forall a.\ ((a \in s) \Rightarrow x) \\
&[\![(\exists a \in s.\ x)]\!]_{st} = ([\![\varphi]\!]_{st})\ \textbf{where} \\
&\qquad\qquad \varphi = \exists a.\ ((a \in s) \wedge x) \\
&[\![(|t| \geqslant n.\ x)]\!]_{((R,r,\Sigma),(\sigma^1,\sigma^2))} = foldcount\ n\ bs\ \textbf{where} \\
&\qquad\qquad\qquad fs = allTups\ r\ \sigma^1\ t \\
&\qquad\qquad\qquad ss = upTups\ fs\ ((R,r,\Sigma),(\bot,\sigma^2)) \\
&\qquad\qquad\qquad bs = satAll\ ss\ x
\end{aligned}
$$

-- small "same function almost everywhere" utility

$up :: (Eq\ a) \Rightarrow (a \rightarrow b) \rightarrow (a,b) \rightarrow (a \rightarrow b)$

$up\ f\ (v,t)\ a = \textbf{if}\ (a \Leftrightarrow v)\ \textbf{then}\ t\ \textbf{else}\ (f\ a)$

-- injection of boolean values into integers

$mb :: \mathbb{B} \rightarrow \mathbb{N}$

$mb\ \textbf{tt} = 1$

$mb\ \textbf{ff} = 0$

-- from a given valuation of tuples tup, db rel of size n and tvar t, computes

-- the list of n different valuation of tuples where t is updated by a tuple from rel

$allTups :: [Tuple\ a] \rightarrow (TVar \rightarrow Tuple\ a) \rightarrow TVar \rightarrow [TVar \rightarrow Tuple\ a]$

$allTups\ r\ \sigma^1\ t = fmap\ (up\ \sigma^1 \circ \lambda y \rightarrow (t,y))\ r$

-- replace a valuation of tuples in a structure by a new one, extended to list

$upTups :: [TVar \rightarrow Tuple\ a] \rightarrow RLInter\ a \rightarrow [RLInter\ a]$

$upTups\ tups\ ((R,r,\Sigma),(\_,\sigma^2)) = fmap\ (\lambda y \rightarrow ((R,r,\Sigma),(y,\sigma^2)))\ tups$

-- the same for attributes

$allAtts :: Sch\ a \rightarrow (AVar \rightarrow a) \rightarrow AVar \rightarrow [AVar \rightarrow a]$

$allAtts\ \Sigma\ \sigma^2\ a = fmap\ (up\ \sigma^2 \circ \lambda y \rightarrow (a,y))\ (\ \Sigma)$

$upAtts :: [AVar \rightarrow a] \rightarrow RLInter\ a \rightarrow [RLInter\ a]$

$upAtts\ atts\ ((R,r,\Sigma),(\sigma^1,\_)) = fmap\ (\lambda y \rightarrow ((R,r,\Sigma),(\sigma^1,y)))\ atts$

-- eval satisfaction on a list of structure

$satAll :: (Eq\ a) \Rightarrow [RLInter\ a] \rightarrow (\mathcal{RL}_a) \rightarrow [\mathbb{B}]$

$satAll\ ss\ x = fmap\ ((flip\ [\![\cdot]\!].)\ x)\ ss$

-- count if at least n values are true

$$
\begin{aligned}
&foldcount :: (Ord\ a, Num\ a) \Rightarrow a \rightarrow [\mathbb{B}] \rightarrow \mathbb{B} \\
&foldcount\ 0\ \_ \quad\ = \textbf{tt} \\
&foldcount\ \_\ [\,] \quad\ = \textbf{ff} \\
&foldcount\ n\ (a : as) = \textbf{if}\ a \\
&\qquad\qquad \textbf{then}\ foldcount\ (n-1)\ as \\
&\qquad\qquad \textbf{else}\ foldcount\ n\ as
\end{aligned}
$$

2.3 Restricted query language

We restrict the language to the class of formulas φ such that :

- *all attribute-variables and tuple-variables are bounded* in φ, so satisfaction $[\![\varphi]\!]_{((R,r,Sigma),(\sigma^1,\sigma^2))}$ is independant of both σ^1 and σ^2. Under Haskell's non-strict semantics, it means that parameters σ^1 and σ^2 won't be evaluated by $[\![\cdot]\!].$, so both can be freely set to $\bot$;

- *φ contains exactly two different schema-variables*, so Σ generally in $S \to 2^R$ is now in $2 \to 2^R \cong (2^R)^2 \cong 2^{2\times R} \cong 2^{R+R} \cong 2^R \times 2^R$ with 2^R encoded with lists.

$isRLQ :: (\mathcal{RL}_a) \to \mathbb{B}$
$isRLQ\ \varphi = \mathbf{let}\ (x, y, z) = FV(\varphi)\ \mathbf{in}\ (size\ x \Leftrightarrow 0) \wedge (size\ y \Leftrightarrow 0) \wedge (size\ z \Leftrightarrow 2)$
$satRLQ :: (Eq\ a) \Rightarrow Sch\ a \to [Tuple\ a] \to ([a], [a]) \to (\mathcal{RL}_a) \to \mathbb{B}$
$satRLQ\ s\ d\ (x, y)\ \varphi \mid (isRLQ\ \varphi) = \mathbf{let}\ (\_, \_, v) = (FV(\varphi))$
$(a : b : [\,]) = toList\ v$
$f = (flip\ up)\ (a, x)\ (up\ \bot\ (b, y))$
$\mathbf{in}\ [\![\varphi]\!]_{((s,d,f),(\bot,\bot))}$
$\mid otherwise\ = error$ `"Formula does not satisfy isRLQ requirement"`

Now we evaluate queries.

$evalQ :: (Eq\ a) \Rightarrow Sch\ a \to [Tuple\ a] \to \mathcal{RL}_a \to [([a], [a])]$
$evalQ\ s\ d\ \varphi = filter\ (\lambda x \to satRLQ\ s\ d\ x\ \varphi)\ (space\ s)$
$space :: Sch\ a \to [([a], [a])]$
$space\ s = \mathbf{let}\ x = (tail \circ subsequences \circ)\ s$
$\mathbf{in}\ (concat \circ fmap\ (\lambda y \to zip\ (repeat\ y)\ x))\ x$

If one can prove that the relation computed by *evalQ* enjoys some nice properties, as for instance closure on Armstrong inference rules provided in [AFP+11], one can use a smarter way to traverse the lattice of pairs of subset of a set R.

Here we naively compute the *complete space* of valuation of schema-variables of size $(2^R - 1)^2$ (we eliminate the $\emptyset$ using *tail*) *then* we evaluate φ instead of pruning the space using rules.

3 Toy sample

3.1 Tuples

Databases:

$db, db', db0 :: [Tuple\ Char]$
$db = [t_1, t_2, t_3, t_4]$
$db' = [t_1, t_2]$
$db0 = fmap\ (mkTuple\ r)\ [[1,3,4],[1,3,4],[2,4,6],[3,5,6]]$
$dba :: [Tuple\ Char]$
$dba = fmap\ (mkTuple\ r')\ [[1,1,0,0],[0,0,1,0],[0,0,0,1],[1,1,1,0],[0,1,0,0]]$
$db_2 :: [Tuple\ Char]$
$db_2 = fmap\ (mkTuple\ r')\ [[0,1,1,1],[0,1,2,1],[0,1,3,2],[1,2,4,2],[1,1,5,3]]$

r	A	B	C
t_1	1	2	1
t_2	1	2	3
t_3	2	2	3
t_4	3	4	5

Table 1: The sample database *db*

r	A	B	C	D
t_1	0	1	1	1
t_2	0	1	2	1
t_3	0	1	3	2
t_4	1	2	4	2
t_4	1	1	5	3

Table 2: The sample database db_2

3.2 Formulas

$(\neq) :: TVar \to TVar \to \mathcal{RL}_a$
$s \neq t = \exists \texttt{"A"}.\ (\neg\ (((s, \texttt{"A"}) = (t, \texttt{"A"}))))$

Formula f_1 characterises functional dependencies between set of attributes X and Y. f_1' is its closure.

$f_1, f_1' :: \mathcal{RL}_{Char}$
$f_1 = \textbf{let}\ fl = \forall \texttt{"A"} \in \texttt{"X"}.\ (((\texttt{"t1"}, \texttt{"A"}) = (\texttt{"t2"}, \texttt{"A"})))$
$\qquad fr = \forall \texttt{"B"} \in \texttt{"Y"}.\ (((\texttt{"t1"}, \texttt{"B"}) = (\texttt{"t2"}, \texttt{"B"})))$
$\quad \textbf{in}\ fl \Rightarrow fr$
$f_1' = (\forall \texttt{"t1"}.\ (\forall \texttt{"t2"}.\ f_1))$

$$
\begin{aligned}
&\mathit{dfCount} :: \mathbb{N} \to \mathcal{RL}_{Char}\\
&\mathit{dfCount}\ n = \textbf{let}\ \mathit{dfl} = \forall\texttt{"A"} \in \texttt{"X"}.\ (((\texttt{"t1"},\texttt{"A"}) = (\texttt{"t2"},\texttt{"A"})))\\
&\qquad\qquad\qquad \mathit{dfr} = \forall\texttt{"B"} \in \texttt{"Y"}.\ (((\texttt{"t1"},\texttt{"B"}) = (\texttt{"t2"},\texttt{"B"})))\\
&\qquad\qquad\qquad \mathit{cnt} = \exists\texttt{"t1"}.\ (|\texttt{"t2"}| \geqslant n.\ \mathit{dfl})\\
&\qquad\qquad\quad \textbf{in}\ (\forall\texttt{"t1"}.\ (\forall\texttt{"t2"}.\ (\mathit{dfl} \Rightarrow \mathit{dfr}))) \wedge \mathit{cnt}
\end{aligned}
$$

Formula g_1 characterises constantness on the A attributes, which is fixed by Σ.

$$
\begin{aligned}
&g_1, g_1' :: \mathcal{RL}_{Char}\\
&g_1 = \forall\texttt{"A"} \in \texttt{"X"}.\ (((\texttt{"s"},\texttt{"A"}) = (\texttt{"t"},\texttt{"A"})))\\
&g_1' = (\forall\texttt{"s"}.\ (\forall\texttt{"t"}.\ g_1))\\
&\mathit{ar}, \mathit{ar}' :: \mathcal{RL}_{Char}\\
&\mathit{ar} = \textbf{let}\ a = \forall\texttt{"A"} \in \texttt{"X"}.\ (((\texttt{"t"},\texttt{"A"}) = 1))\\
&\qquad\qquad b = \forall\texttt{"B"} \in \texttt{"Y"}.\ (((\texttt{"t"},\texttt{"B"}) = 1))\\
&\qquad \textbf{in}\ \ a \Rightarrow b\\
&\mathit{ar}' = (\forall\texttt{"t"}.\ \mathit{ar})\\
&\mathit{nar}, \mathit{nar}' :: \mathcal{RL}_{Char}\\
&\mathit{nar} = \textbf{let}\ a = \forall\texttt{"A"} \in \texttt{"X"}.\ (((\texttt{"t"},\texttt{"A"}) = 1))\\
&\qquad\qquad b = \forall\texttt{"B"} \in \texttt{"Y"}.\ (((\texttt{"t"},\texttt{"B"}) = 0))\\
&\qquad \textbf{in}\ \ a \Rightarrow b\\
&\mathit{nar}' = (\forall\texttt{"t"}.\ \mathit{nar})\\
&\mathit{countA1} :: \mathbb{N} \to \mathcal{RL}_{Char}\\
&\mathit{countA1}\ n = \textbf{let}\ a = \forall\texttt{"A"} \in \texttt{"X"}.\ (((\texttt{"t"},\texttt{"A"}) = 1))\\
&\quad \textbf{in} \qquad\qquad (|\texttt{"t"}| \geqslant n.\ a)\\
&h, h' :: \mathcal{RL}_{Char}\\
&h = \textbf{let}\ a = \forall\texttt{"A"} \in \texttt{"X"}.\ (((\texttt{"t1"},\texttt{"A"}) = (\texttt{"t2"},\texttt{"A"})) \wedge (\neg \circ \cdot)\ ((\texttt{"t3"},\texttt{"A"}) = (\texttt{"t4"},\texttt{"A"})))\\
&\qquad\quad b = \forall\texttt{"B"} \in \texttt{"Y"}.\ (((\texttt{"t1"},\texttt{"B"}) = (\texttt{"t2"},\texttt{"B"})) \wedge (\neg \circ \cdot)\ ((\texttt{"t3"},\texttt{"B"}) = (\texttt{"t4"},\texttt{"B"})))\\
&\quad \textbf{in}\quad a \Rightarrow b\\
&h' = (\forall\texttt{"t1"}.\ (\forall\texttt{"t2"}.\ (\forall\texttt{"t3"}.\ (\forall\texttt{"t4"}.\ h))))\\
&\mathit{dfmin}, \mathit{dfmin}' :: \mathcal{RL}_{Char}\\
&\mathit{dfmin} = \textbf{let}\ \mathit{fl} = \forall\texttt{"A"} \in \texttt{"X"}.\ (((\texttt{"t1"},\texttt{"A"}) = (\texttt{"t2"},\texttt{"A"})))\\
&\qquad\qquad\quad \mathit{fr} = \forall\texttt{"B"} \in \texttt{"Y"}.\ (((\texttt{"t1"},\texttt{"B"}) = (\texttt{"t2"},\texttt{"B"})))\\
&\qquad\qquad\quad \mathit{fd} = \forall\texttt{"A"} \in \texttt{"X"}.\ (\forall\texttt{"B"} \in \texttt{"Y"}.\ (\neg\ ((\texttt{"A"} = \texttt{"B"}))))\\
&\qquad\qquad\quad \mathit{fu} = \forall\texttt{"B1"} \in \texttt{"Y"}.\ (\forall\texttt{"B2"} \in \texttt{"Y"}.\ ((\texttt{"B1"} = \texttt{"B2"})))\\
&\qquad\quad \textbf{in}\ (\mathit{fl} \Rightarrow \mathit{fr}) \wedge \mathit{fd} \wedge \mathit{fu}\\
&\mathit{dfmin}' = (\forall\texttt{"t1"}.\ (\forall\texttt{"t2"}.\ \mathit{dfmin}))
\end{aligned}
$$

So far we have :

- $FV(f_1)$ = (*fromList* ["t1","t2"],*fromList* [],*fromList* ["X","Y"])
- $FV(f_1')$ = (*fromList* [],*fromList* [],*fromList* ["X","Y"])
- $FV(g_1)$ = (*fromList* ["s","t"],*fromList* [],*fromList* ["X"])
- $FV(g_1')$ = (*fromList* [],*fromList* [],*fromList* ["X"])

3.3 Structures

$$
\begin{aligned}
&\Sigma_1, \Sigma_2 :: \mathit{SVar} \to [\mathit{Char}]\\
&\Sigma_1\ \texttt{"X"} = \texttt{"A"}\\
&\Sigma_1\ \texttt{"Y"} = \texttt{"C"}\\
&\Sigma_1\ \_ = \bot
\end{aligned}
$$

Σ_2 "X" = "AC"
Σ_2 "Y" = "B"
Σ_2 _ = $\perp$
$\sigma^1_1, \sigma^1_2 :: TVar \to Tuple\ Char$
σ^1_1 "t" = t_1
σ^1_1 "s" = t_2
σ^1_1 _ = $\perp$
σ^1_2 "t" = t_0
σ^1_2 "s" = t_1
σ^1_2 _ = $\perp$

3.4 Satisfaction

We evaluate:

- g_1 on $((r, db, \Sigma_1), (\sigma^1_1, \perp))$: **tt**
- g_1 on $((r, db, \Sigma_1), (\sigma^1_1, \perp))$: **ff**
- g_1 on $((r, db, \Sigma_1), (\sigma^1_2, \perp))$: **ff**
- g_1 on $((r, db, \Sigma_2), (\sigma^1_2, \perp))$: **ff**
- g_1 on $((r, db', \Sigma_1), (\sigma^1_1, \perp))$: **tt**
- g_1 on $((r, db', \Sigma_1), (\sigma^1_1, \perp))$: **ff**
- g_1 on $((r, db', \Sigma_1), (\sigma^1_2, \perp))$: **ff**
- g_1 on $((r, db', \Sigma_2), (\sigma^1_2, \perp))$: **ff**
- g'_1 on $((r, db, \Sigma_1), (\sigma^1_1, \perp))$: **ff**
- g'_1 on $((r, db, \Sigma_1), (\sigma^1_1, \perp))$: **ff**
- g'_1 on $((r, db, \Sigma_1), (\sigma^1_2, \perp))$: **ff**
- g'_1 on $((r, db, \Sigma_2), (\sigma^1_2, \perp))$: **ff**
- g'_1 on $((r, db', \Sigma_1), (\sigma^1_1, \perp))$: **tt**
- g'_1 on $((r, db', \Sigma_1), (\sigma^1_1, \perp))$: **ff**
- g'_1 on $((r, db', \Sigma_1), (\sigma^1_2, \perp))$: **tt**
- g'_1 on $((r, db', \Sigma_2), (\sigma^1_2, \perp))$: **ff**
- f_1 on $((r, db, \Sigma_1), (\sigma^1_1, \perp))$: $***Exception : Prelude.\perp$
- f_1 on $((r, db, \Sigma_1), (\sigma^1_1, \perp))$: $***Exception : Prelude.\perp$
- f_1 on $((r, db, \Sigma_1), (\sigma^1_2, \perp))$: $***Exception : Prelude.\perp$
- f_1 on $((r, db, \Sigma_2), (\sigma^1_2, \perp))$: $***Exception : Prelude.\perp$
- f_1 on $((r, db', \Sigma_1), (\sigma^1_1, \perp))$: $***Exception : Prelude.\perp$
- f_1 on $((r, db', \Sigma_1), (\sigma^1_1, \perp))$: $***Exception : Prelude.\perp$
- f_1 on $((r, db', \Sigma_1), (\sigma^1_2, \perp))$: $***Exception : Prelude.\perp$
- f_1 on $((r, db', \Sigma_2), (\sigma^1_2, \perp))$: $***Exception : Prelude.\perp$
- f'_1 on $((r, db, \Sigma_1), (\sigma^1_1, \perp))$: **ff**
- f'_1 on $((r, db, \Sigma_1), (\sigma^1_1, \perp))$: **tt**
- f'_1 on $((r, db, \Sigma_1), (\sigma^1_2, \perp))$: **ff**
- f'_1 on $((r, db, \Sigma_2), (\sigma^1_2, \perp))$: **tt**
- f'_1 on $((r, db', \Sigma_1), (\sigma^1_1, \perp))$: **ff**
- f'_1 on $((r, db', \Sigma_1), (\sigma^1_1, \perp))$: **tt**
- f'_1 on $((r, db', \Sigma_1), (\sigma^1_2, \perp))$: **ff**
- f'_1 on $((r, db', \Sigma_2), (\sigma^1_2, \perp))$: **tt**

3.5 Queries

We specialize on $\mathcal{RL}$-queries, we evaluate *evalQ r db* f_1', its length is 27:

$\{A \to A, A \to B, A \to AB, B \to B, AB \to A, AB \to B, AB \to AB, C \to B, C \to C, C \to BC, AC \to A, AC \to B, AC \to AB, AC \to C, AC \to AC, AC \to BC, AC \to ABC, BC \to B, BC \to C, BC \to BC, ABC \to A, ABC \to B, ABC \to AB, ABC \to C, ABC \to AC, ABC \to BC, ABC \to ABC\}$

Now evaluate *evalQ r db0 h'*, whose length is 31:

$\{A \to A, A \to B, A \to AB, B \to A, B \to B, B \to AB, AB \to A, AB \to B, AB \to AB, C \to C, AC \to A, AC \to B, AC \to AB, AC \to C, AC \to AC, AC \to BC, AC \to ABC, BC \to A, BC \to B, BC \to AB, BC \to C, BC \to AC, BC \to BC, BC \to ABC, ABC \to A, ABC \to B, ABC \to AB, ABC \to C, ABC \to AC, ABC \to BC, ABC \to ABC\}$

3.6 Queries with support

We are interested into formulas *dfCount n*. First we check whether *evalQ r db* $f_1' \Leftrightarrow$ *evalQ r db* (*dfCount* 0) **tt**. Then we evaluate *dfCount n* for increasing values of *n* :

1. *evalQ r db* (*dfCount* 1) (27 results) : $\{A \to A, A \to B, A \to AB, B \to B, AB \to A, AB \to B, AB \to AB, C \to B, C \to C, C \to BC, AC \to A, AC \to B, AC \to AB, AC \to C, AC \to AC, AC \to BC, AC \to ABC, BC \to B, BC \to C, BC \to BC, ABC \to A, ABC \to B, ABC \to AB, ABC \to C, ABC \to AC, ABC \to BC, ABC \to ABC\}$
2. *evalQ r db* (*dfCount* 2) (13 results) : $\{A \to A, A \to B, A \to AB, B \to B, AB \to A, AB \to B, AB \to AB, C \to B, C \to C, C \to BC, BC \to B, BC \to C, BC \to BC\}$
3. *evalQ r db* (*dfCount* 3) (1 results) : $\{B \to B\}$
4. *evalQ r db* (*dfCount* 4) (0 results) : $\{\}$

Now, something ugly with *df3* and *df3c*, the *functional dependencies with 3 different tuples* :

$df3, df3' :: \mathcal{RL}_{Char}$
$df3 = \textbf{let}\ a = \forall \texttt{"A"} \in \texttt{"X"}.\ (((\texttt{"t1"}, \texttt{"A"}) = (\texttt{"t2"}, \texttt{"A"})) \land ((\texttt{"t2"}, \texttt{"A"}) = (\texttt{"t3"}, \texttt{"A"})) \land d)$
$\qquad d = (\texttt{"t1"} \neq \texttt{"t2"}) \land (\texttt{"t2"} \neq \texttt{"t3"}) \land (\texttt{"t1"} \neq \texttt{"t3"})$
$\qquad b = \forall \texttt{"B"} \in \texttt{"Y"}.\ (((\texttt{"t1"}, \texttt{"B"}) = (\texttt{"t2"}, \texttt{"B"})) \land ((\texttt{"t2"}, \texttt{"B"}) = (\texttt{"t3"}, \texttt{"B"})))$
$\quad \textbf{in}\quad a \Rightarrow b$
$df3' = (\forall \texttt{"t1"}.\ (\forall \texttt{"t2"}.\ (\forall \texttt{"t3"}.\ (df3))))$
$df3c :: \mathbb{N} \to \mathcal{RL}_{Char}$
$df3c\ n = \textbf{let}\ a = \forall \texttt{"A"} \in \texttt{"X"}.\ (((\texttt{"t1"}, \texttt{"A"}) = (\texttt{"t2"}, \texttt{"A"})) \land ((\texttt{"t2"}, \texttt{"A"}) = (\texttt{"t3"}, \texttt{"A"})) \land d)$
$\qquad d = (\texttt{"t1"} \neq \texttt{"t2"}) \land (\texttt{"t2"} \neq \texttt{"t3"}) \land (\texttt{"t1"} \neq \texttt{"t3"})$
$\qquad b = \forall \texttt{"B"} \in \texttt{"Y"}.\ (((\texttt{"t1"}, \texttt{"B"}) = (\texttt{"t2"}, \texttt{"B"})) \land ((\texttt{"t2"}, \texttt{"B"}) = (\texttt{"t3"}, \texttt{"B"})))$
$\qquad c = \exists \texttt{"t1"}.\ (\exists \texttt{"t2"}.\ (|\texttt{"t3"}| \geqslant n.\ a))$
$\qquad \textbf{in}\ (\forall \texttt{"t1"}.\ (\forall \texttt{"t2"}.\ (\forall \texttt{"t3"}.\ (a \Rightarrow b)))) \land c$

- *evalQ* r' db_2 $df3'$ has 187 results,
- *evalQ* r' db_2 f_1' has 147 results,
- the difference is $\{A \to AB, A \to B, ABD \to ABC, ABD \to ABCD, ABD \to AC, ABD \to ACD, ABD \to BC, ABD \to BCD, ABD \to C, ABD \to CD, AD \to ABC, AD \to ABCD, AD \to$

$AC, AD \rightarrow ACD, AD \rightarrow BC, AD \rightarrow BCD, AD \rightarrow C, AD \rightarrow CD, BD \rightarrow ABC, BD \rightarrow ABCD, BD \rightarrow AC, BD \rightarrow ACD, BD \rightarrow BC, BD \rightarrow BCD, BD \rightarrow C, BD \rightarrow CD, D \rightarrow A, D \rightarrow AB, D \rightarrow ABC, D \rightarrow ABCD, D \rightarrow ABD, D \rightarrow AC, D \rightarrow ACD, D \rightarrow AD, D \rightarrow B, D \rightarrow BC, D \rightarrow BCD, D \rightarrow BD, D \rightarrow C, D \rightarrow CD\}$

1. *evalQ* r' db_2 (*df3c* 1) has 7 results : $\{A \rightarrow A, A \rightarrow B, A \rightarrow AB, B \rightarrow B, AB \rightarrow A, AB \rightarrow B, AB \rightarrow AB\}$

2. *evalQ* r' db_2 (*df3c* 2) has 1 results : $\{B \rightarrow B\}$

References

[AFP+11] Marie Agier, Christine Froidevaux, Jean-Marc Petit, Yoan Renaud, and Jef Wijsen. On armstrong-compliant logical query languages. In *4th International Workshop on Logic in Databases (LID 2011)*, March 2011. collocated with the 2011 EDBT/ICDT conference.

[Knu84] Donald E. Knuth. Literate programming. *Comput. J.*, 27(2):97–111, 1984.

[PJHA+99] Simon L Peyton Jones, John Hughes, Lennart Augustsson, Dave Barton, Brian Boutel, Warren Burton, Joseph Fasel, Kevin Hammond, Ralf Hinze, Paul Hudak, Thomas Johnsson, Mark Jones, John Launchbury, Erik Meijer, John Peterson, Alastair Reid, Colin Runciman, and Philip Wadler. The haskell 98 report, 1999. Available from `http://www.haskell.org/onlinereport/`.