A Query Language Based on the Ambient Logic

Luca Cardelli

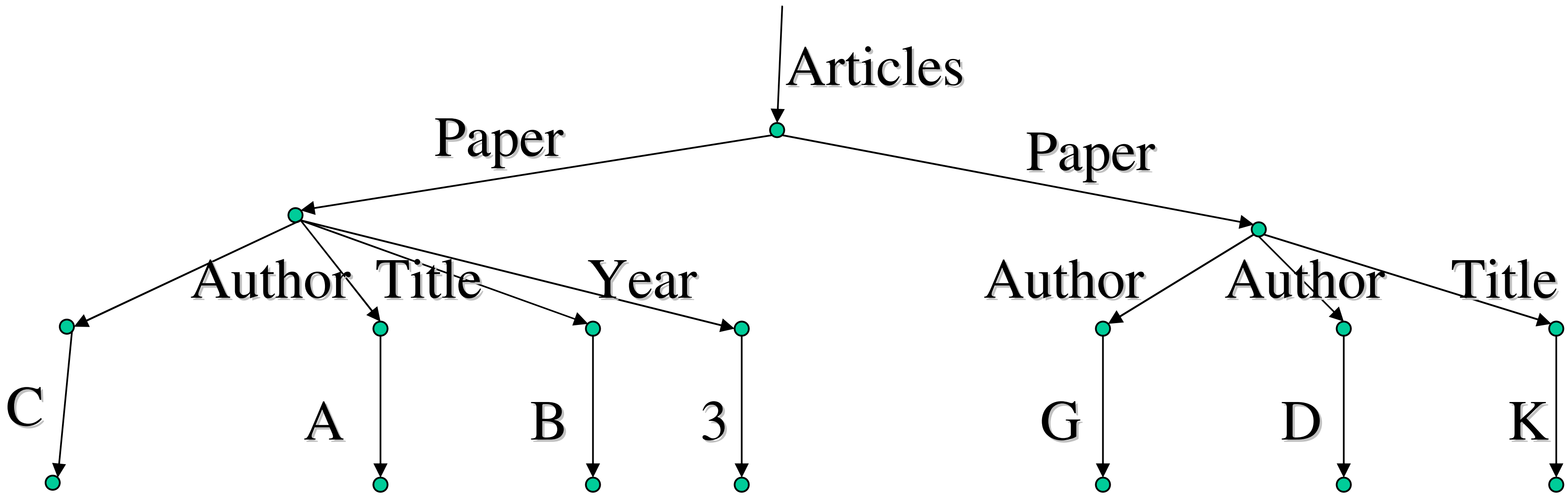
Microsoft Research

Giorgio Ghelli

University of Pisa

2001-07-06 Concoord - Lipari

Semistructured Data



- A tree (or graph), unordered (or ordered).
- Invented for “flexible” data representation (just like S-expressions…) for semi-irregular data like address books and bibliographies.
- Adopted by the DB community for the purpose of merging databases from different uncoordinated sources (without a common schema). Typically, web data that belongs to different institutions.

Unusual Data

- Not really arrays/lists:
  - Many children with the same label, instead of indexed children.
  - Mixture of repeated and non repeated labels under a node.
- Not really records:
  - Many children with the same label.
  - Missing/additional fields with no tagging information.
- Not really variants:
  - Labeled but untagged unions.
- Unusual data.
  - Yet, it aims to be the new universal standard for interoperability of programming languages, databases, e-commerce...

Unusual Data Manipulation

- New "flexible" type theories are required.
  - Based on the "effects" of processes over trees (Ambient Types).
  - Based on tree automata (Xduce).
- New processing languages required.
  - Xduce.
  - Various web scripting abominations.
- New query languages required. Various approaches.
  - From simple: Query existence of paths through the tree.
  - To fuzzy: Query whether a tree is kind of similar to another one.
  - To fancy: Query whether a tree is produced by a tree grammar.
  - To popular: "SQL for trees/graphs".

Analogies

- An accidental(?) similarity between two areas:
- Semistructured Data is the way it is because:
  - *"You cannot rely on uniform structure"* of data.
    Must abandon schemas based on records and tables.
  - Adopt "self-describing" data structures:
    Use edge-labeled trees (or graphs).
- Mobile Computation is the way it is because:
  - *"You cannot rely on static structure"* of networks.
    Must abandon type systems based on records and disjoint unions.
  - Adopt "self-describing" network structures:
    Use edge-labeled trees (or graphs) of locations and agents.
- Both arose out of the Internet, because things there are just too dynamic for traditional notions of data and computation.

Relevance to Concurrency and Coordination

- Immediate implication: a new, uniform, model of data and computation on the Web, with opportunities for cross-fertilization:
  - Programming technology can be used to typecheck, navigate, and transform both dynamic network structures and the semistructured data they contain. Uniformly.
  - Database technology can be used to search through both dynamic network structures ("resource discovery"), and the semistructured data they contain. Uniformly.
- This convergence is still a dream, but it did motivate us to apply a particular technology developed for mobile computation to semistructured data:
  - Specification Logic ➔ Query Logic

Concepts

- *Information trees* $I \in \mathcal{FT}$ (semistructured data)
- *Information terms* F (denoting information trees)
- *Formulas* $\mathcal{A}$ (denoting sets of information trees)
- A semantics of terms $[\![F]\!] \in \mathcal{FT}$
- A semantics of formulas $[\![\mathcal{A}]\!] \subseteq \mathcal{FT}$
- A satisfaction (i.e. matching) relation $F \vDash \mathcal{A}$ (i.e. $[\![F]\!] \in [\![\mathcal{A}]\!]$)
- A *query language* Q (including *from* $F \vDash \mathcal{A}$ *select* Q')
- A (naïve) query semantics $[\![Q]\!] \in \mathcal{FT}$
- A *table algebra* for matching evaluation (i.e. for $F \vDash \mathcal{A}$)
- A (refined) query semantics / query evaluation procedure for Q, based on the table algebra. Correct w.r.t. $[\![Q]\!]$.

Semantics: Information Trees

- Our semantic model for semistructured data: unordered edge-labeled finite-depth trees.
- Just to make it precise:

  Λ is a countable collection of *labels*; m, n, … .

  - $\mathcal{IT}$ is the collection of *information trees* I:
    - The empty multiset, $\{\}^+$ is in $\mathcal{IT}$.
      A root.
    - If m is in Λ and I is in $\mathcal{IT}$,
      then the singleton multiset $\{\langle m, I\rangle\}^+$ is in $\mathcal{IT}$.
      An edge labeled m, leading to I.
    - $\mathcal{IT}$ is closed under multiset union $\bigcup^+_{j\in J} I_j$,
      where J is a (possibly infinite) index set.
      The root-merge of all the I_j.

Syntax: Information Terms

F ::=

$\mathbf{0}$	denoting the empty multiset
$m[F]$	denoting a singleton multiset
$F \mid F$	denoting binary multiset union

$[\![\mathbf{0}]\!]$	$\triangleq$	$\mathbf{0}$	where	$\mathbf{0}$	$\triangleq$	$\{\,\}^+$
$[\![m[F]]\!]$	$\triangleq$	$m[[\![F]\!]]$	where	$m[I]$	$\triangleq$	$\{\langle m, I\rangle\}^+$
$[\![F' \mid F'']\!]$	$\triangleq$	$[\![F']\!] \mid [\![F'']\!]$	where	$I' \mid I''$	$\triangleq$	$I' \cup^+ I''$

- Often, $m[\mathbf{0}]$ is written $m[]$, or simply m.
- Define an equivalence $\equiv$ such that $F \equiv F'$ iff $[\![F]\!] = [\![F']\!]$.

Equivalence of Information Term

$F \equiv F$

$F \equiv F' \;\Rightarrow\; F' \equiv F$

$F \equiv F',\ F' \equiv F'' \;\Rightarrow\; F \equiv F''$

$F \equiv F' \;\Rightarrow\; m[F] \equiv m[F']$

$F \equiv F' \;\Rightarrow\; F \mid F'' \equiv F' \mid F''$

$F \mid \mathbf{0} \equiv F$

$F \mid F' \equiv F' \mid F$

$(F \mid F') \mid F'' \equiv F \mid (F' \mid F'')$

N.B.: $F \mid F' \not\equiv F$ (these are multisets)

$$F \equiv F' \quad \text{iff} \quad [\![F]\!] = [\![F']\!]$$

Example

```
Articles[
    Paper[
        Author[Cardelli] | Author[Gordon] |
        Title [Anywhere] |
        Year[2000] |
        Conf[POPL]
        ] |
    Paper[
        Author[Ghelli] |
        Title[Recursive] |
        Proceedings[VLDB] |
        Year[1998] |
        Editor[SV]
        ]
    ]
```

The Query Logic

$\mathcal{A}, \mathcal{B} \in \Phi$ ::=	Formulas	(η is a name n or a variable x)
$\mathbf{T}$	true	
$\neg\mathcal{A}$	negation	
$\mathcal{A} \wedge \mathcal{B}$	conjunction	
$\exists x.\mathcal{A}$	existential quantification over label variables	
$\eta \sim \eta'$	label comparison (e.g. equality, regexp match..)	
$\mathbf{0}$	root	
$\eta[\mathcal{A}]$	edge	
$\mathcal{A} \mid \mathcal{B}$	composition	
X	tree variable	
$\exists X.\mathcal{A}$	existential quantification over tree variables	
ξ	recursion variable	
$\mu\xi.\mathcal{A}$	recursive formula (least fixpoint) ξ may occur only *positively* in $\mathcal{A}$	

Satisfaction (first try)

- “Match a database F to a query $\mathcal{A}$, collect matches in ρ.”

$$
\begin{array}{lcll}
F \vDash_\rho \mathbf{T} & & & \text{“success”, but no further info} \\
F \vDash_\rho \neg\mathcal{A} & \Leftrightarrow & \neg F \vDash_\rho \mathcal{A} & \\
F \vDash_\rho \mathcal{A} \wedge \mathcal{B} & \Leftrightarrow & F \vDash_\rho \mathcal{A} \wedge F \vDash_\rho \mathcal{B} & \\
F \vDash_\rho \exists x.\mathcal{A} & \Leftrightarrow & \exists n \in \Lambda.\ F \vDash_\rho \mathcal{A}\{x \leftarrow n\} & \\
F \vDash_\rho n \sim m & \Leftrightarrow & n \sim m & \\
F \vDash_\rho \mathbf{0} & \Leftrightarrow & F \equiv \mathbf{0} & \\
F \vDash_\rho n[\mathcal{A}] & \Leftrightarrow & \exists F'.\ F \equiv n[F'] \wedge F' \vDash_\rho \mathcal{A} & \\
F \vDash_\rho \mathcal{A} \mid \mathcal{B} & \Leftrightarrow & \exists F', F''.\ F \equiv F' \mid F'' \wedge F' \vDash_\rho \mathcal{A} \wedge F'' \vDash_\rho \mathcal{B} & \\
F \vDash_\rho X & \Leftrightarrow & F \equiv \rho(X) & \\
F \vDash_\rho \exists X.\mathcal{A} & \Leftrightarrow & \exists F'.\ F \vDash_{\rho[X \mapsto F']} \mathcal{A} & (\mathcal{A}\{X \leftarrow F'\} \text{ bogus}) \\
F \vDash_\rho \mu\xi.\mathcal{A} & \Leftrightarrow & F \vDash_\rho \mathcal{A}\{\xi \leftarrow \mu\xi.\mathcal{A}\} & \text{a match}
\end{array}
$$

- However, this is not a well-formed definition of $F \vDash_\rho \mathcal{A}$, because the μ case is circular. We need a proper semantics of fixpoints.

Example: Search

- Search:
  - "Find one of my articles (ignore non-articles); bind to X all info under the *article* label":

    $$S \;=\; \exists X.\ article[(author[Cardelli] \mid \mathbf{T}) \wedge X] \mid \mathbf{T}$$

  - Can use recursive formulas to search deeper:

    $$\mu\xi.\ S \vee \exists x.\ (x[\xi] \mid \mathbf{T})$$

- Not a query language yet.
  - It searches for one instance, not all instances.
  - Some *collecting* primitive must be added. This is going to be based on the logical notion of *satisfaction*.
- N.B.: $\exists X.\mathcal{A}$ is, logically, a bit strange:
  - X denotes a *singleton* set of trees (the match), not a general set of trees like a regular formula.

Example: Path Expressions

- Various kinds of path expressions can be defined:

$pq[\mathcal{A}]$	$\triangleq$	$p[q[\mathcal{A}]]$	path concatenation
$p^*[\mathcal{A}]$	$\triangleq$	$\mu\xi.\, \mathcal{A} \vee p[\xi]$	path iteration
$(p \vee q)[\mathcal{A}]$	$\triangleq$	$p[\mathcal{A}] \vee q[\mathcal{A}]$	path disjunction
$p(X)[\mathcal{A}]$	$\triangleq$	$p[X \wedge \mathcal{A}]$	binding the tree at the end
$.\eta[\mathcal{A}]$	$\triangleq$	$(\exists x.\, x{\sim}\eta \wedge x[\mathcal{A}]) \mid \mathbf{T}$	some η leads to $\mathcal{A}$
$.\neg\eta[\mathcal{A}]$	$\triangleq$	$(\exists x.\, \neg x{\sim}\eta \wedge x[\mathcal{A}]) \mid \mathbf{T}$	some non-η leads to $\mathcal{A}$
$!\eta[\mathcal{A}]$	$\triangleq$	$(\forall x.\, x{\sim}\eta \Rightarrow x[\Rightarrow\mathcal{A}]) \parallel \mathbf{F}$	all η lead to $\mathcal{A}$
etc.			

- "*X* is an article that deals with *SSD*, or from each one can reach, through citations, an article that deals with *SSD*":

$.article(X)(.cites.article)^*.keyword[SSD]$

Example: Schemas

- A logic is a "very rich type system". Hence we can comfortably represent various kinds of schemas.
  - However, extensions (or unpleasant encodings) are required for ordered data: $\mathcal{A} \mid \mathcal{B}$ vs. $\mathcal{A} ; \mathcal{B}$.
- Ex.: Xduce-like schemas:

$\mathbf{0}$	the empty tree	
$\mathcal{A} \mid \mathcal{B}$	an $\mathcal{A}$ next to a $\mathcal{B}$	
$\mathcal{A} \vee \mathcal{B}$	either an $\mathcal{A}$ or a $\mathcal{B}$	
$n[\mathcal{A}]$	an edge n leading to an $\mathcal{A}$	
$\mathcal{A}^*$	$\triangleq \mu\xi.\mathbf{0} \vee (\mathcal{A} \mid \xi)$	the merge of zero or more $\mathcal{A}$s
$\mathcal{A}^+$	$\triangleq \mathcal{A} \mid \mathcal{A}^*$	the merge of one or more $\mathcal{A}$s
$\mathcal{A}^?$	$\triangleq \mathbf{0} \vee \mathcal{A}$	zero or one $\mathcal{A}$

Semantics

$$\llbracket \mathbf{T} \rrbracket_{\rho,\delta} \triangleq \mathcal{FT}$$

$$F \vDash_{\rho,\delta} \mathcal{A} \triangleq \llbracket F \rrbracket \in \llbracket \mathcal{A} \rrbracket_{\rho,\delta}$$

$$\llbracket \neg\mathcal{A} \rrbracket_{\rho,\delta} \triangleq \mathcal{FT} \setminus \llbracket \mathcal{A} \rrbracket_{\rho,\delta}$$

$$\llbracket \mathcal{A} \wedge \mathcal{B} \rrbracket_{\rho,\delta} \triangleq \llbracket \mathcal{A} \rrbracket_{\rho,\delta} \cap \llbracket \mathcal{B} \rrbracket_{\rho,\delta}$$

$$\llbracket \exists x.\mathcal{A} \rrbracket_{\rho,\delta} \triangleq \bigcup_{n\in\Lambda} \llbracket \mathcal{A} \rrbracket_{\rho[x\mapsto n],\delta}$$

$$\llbracket \eta \sim \eta' \rrbracket_{\rho,\delta} \triangleq \textit{if } \rho(\eta) \sim \rho(\eta') \textit{ then } \mathcal{FT} \textit{ else } \{\}$$

$$\llbracket \mathbf{0} \rrbracket_{\rho,\delta} \triangleq \{\mathbf{0}\}$$

$$\llbracket \eta[\mathcal{A}] \rrbracket_{\rho,\delta} \triangleq \{\rho(\eta)[I] \parallel I \in \llbracket \mathcal{A} \rrbracket_{\rho,\delta}\}$$

$$\llbracket \mathcal{A} \mid \mathcal{B} \rrbracket_{\rho,\delta} \triangleq \{I \mid I' \parallel I \in \llbracket \mathcal{A} \rrbracket_{\rho,\delta} \wedge I' \in \llbracket \mathcal{B} \rrbracket_{\rho,\delta}\}$$

$$\llbracket X \rrbracket_{\rho,\delta} \triangleq \{\rho(X)\}$$

$$\llbracket \exists X.\mathcal{A} \rrbracket_{\rho,\delta} \triangleq \bigcup_{I\in\mathcal{FT}} \llbracket \mathcal{A} \rrbracket_{\rho[X\mapsto I],\delta}$$

$$\llbracket \xi \rrbracket_{\rho,\delta} \triangleq \delta(\xi)$$

$$\llbracket \mu\xi.\mathcal{A} \rrbracket_{\rho,\delta} \triangleq \bigcap \{S \subseteq \mathcal{FT} \parallel S \supseteq \llbracket \mathcal{A} \rrbracket_{\rho,\delta[\xi\mapsto S]}\}$$

$$\llbracket\ \rrbracket \in \Phi \times ((Nvar \cup \Lambda \to \Lambda) \cup (Tvar \to \mathcal{FT})) \times (Rvar \to \mathcal{P}(\mathcal{FT})) \to \mathcal{P}(\mathcal{FT})$$

Derived Operators

$\mathbf{F}$	$\triangleq$	$\neg\mathbf{T}$	falsity
$\mathcal{A} \vee \mathcal{B}$	$\triangleq$	$\neg(\neg\mathcal{A} \wedge \neg\mathcal{B})$	disjunction
$\eta \not\sim \eta'$	$\triangleq$	$\neg(\eta \sim \eta')$	label diversity
$\forall x.\mathcal{A}$	$\triangleq$	$\neg\exists x.\neg\mathcal{A}$	universal name quantification
$\forall X.\mathcal{A}$	$\triangleq$	$\neg\exists X.\neg\mathcal{A}$	universal tree quantification
$\nu\xi.\mathcal{A}$	$\triangleq$	$\neg\mu\xi.\neg\mathcal{A}\{\xi\leftarrow\neg\xi\}$	maximal fixpoint
$\mathbf{1}$	$\triangleq$	$\neg\mathbf{0}$	non-empty
$\eta[\Rightarrow\mathcal{A}]$	$\triangleq$	$\neg(\eta[\neg\mathcal{A}])$	edge implication
			if edge is η, it leads to $\mathcal{A}$
$\mathcal{A} \parallel \mathcal{B}$	$\triangleq$	$\neg(\neg\mathcal{A} \mid \neg\mathcal{B})$	decomposition

for every partition, either $\mathcal{A}$ holds one part, or $\mathcal{B}$ holds on the other

($\mathcal{A} \parallel \mathbf{F}$: all partitions satisfy $\mathcal{A}$)

Dualization

- We can dualize all operators.

$$\neg\mathbf{0} \rightsquigarrow \mathbf{1} \qquad \neg\mathbf{1} \rightsquigarrow \mathbf{0}$$

$$\neg\eta[\mathcal{A}] \rightsquigarrow \eta[\Rightarrow\neg\mathcal{A}] \qquad \neg\eta[\Rightarrow\mathcal{A}] \rightsquigarrow \eta[\neg\mathcal{A}]$$

$$\neg(\mathcal{A} \mid \mathcal{B}) \rightsquigarrow \neg\mathcal{A} \parallel \neg\mathcal{B} \qquad \neg(\mathcal{A} \parallel \mathcal{B}) \rightsquigarrow \neg\mathcal{A} \mid \neg\mathcal{B}$$

$$\neg\nu\xi.\mathcal{A} \rightsquigarrow \mu\xi.\neg\mathcal{A}\{\xi\leftarrow\neg\xi\} \qquad \neg\mu\xi.\mathcal{A} \rightsquigarrow \nu\xi.\neg\mathcal{A}\{\xi\leftarrow\neg\xi\}$$

  - Plus the usual DeMorgan laws.
- This gives us an implementation strategy for $\neg\mathcal{A}$.
  - If we take the dual operators as primitive and implement them directly, we can then “push negation to the leaves”.

Logical Equations

- We can commute/distribute/simplify many operators.
  - This gives us opportunities for query optimization.

$\eta[\mathcal{A}]$	$\Leftrightarrow$	$\eta[\mathbf{T}] \wedge \eta[\Rightarrow\mathcal{A}]$	$\eta[\Rightarrow\mathcal{A}]$	$\Leftrightarrow$	$\eta[\mathbf{T}] \Rightarrow \eta[\mathcal{A}]$
$\eta[\mathbf{F}]$	$\Leftrightarrow$	$\mathbf{F}$	$\eta[\Rightarrow\mathbf{T}]$	$\Leftrightarrow$	$\mathbf{T}$
$\eta[\mathcal{A} \wedge \mathcal{B}]$	$\Leftrightarrow$	$\eta[\mathbf{T}] \wedge \eta[\mathcal{B}]$	$\eta[\Rightarrow\mathcal{A} \vee \mathcal{B}]$	$\Leftrightarrow$	$\eta[\Rightarrow\mathcal{A}] \vee \eta[\Rightarrow\mathcal{B}]$
$\eta[\mathcal{A} \vee \mathcal{B}]$	$\Leftrightarrow$	$\eta[\mathcal{A}] \vee \eta[\mathcal{B}]$	$\eta[\Rightarrow\mathcal{A} \wedge \mathcal{B}]$	$\Leftrightarrow$	$\eta[\Rightarrow\mathcal{A}] \wedge \eta[\Rightarrow\mathcal{B}]$
$\mathcal{A} \mid \mathbf{F}$	$\Leftrightarrow$	$\mathbf{F}$	$\mathcal{A} \parallel \mathbf{T}$	$\Leftrightarrow$	$\mathbf{T}$
$\mathbf{T} \mid \mathbf{T}$	$\Leftrightarrow$	$\mathbf{T}$	$\mathbf{F} \parallel \mathbf{F}$	$\Leftrightarrow$	$\mathbf{F}$
$\mathcal{A} \mid \mathbf{0}$	$\Leftrightarrow$	$\mathcal{A}$	$\mathcal{A} \parallel \mathbf{1}$	$\Leftrightarrow$	$\mathcal{A}$
$\mathcal{A} \mid \mathcal{B}$	$\Leftrightarrow$	$\mathcal{B} \mid \mathcal{A}$	$\mathcal{A} \parallel \mathcal{B}$	$\Leftrightarrow$	$\mathcal{B} \parallel \mathcal{A}$
$(\mathcal{A} \mid \mathcal{B}) \mid \mathcal{C}$	$\Leftrightarrow$	$\mathcal{A} \mid (\mathcal{B} \mid \mathcal{C})$	$(\mathcal{A} \parallel \mathcal{B}) \parallel \mathcal{C}$	$\Leftrightarrow$	$\mathcal{A} \parallel (\mathcal{B} \parallel \mathcal{C})$
$\mathcal{A} \mid (\mathcal{B} \vee \mathcal{C})$	$\Leftrightarrow$	$(\mathcal{A} \mid \mathcal{B}) \vee (\mathcal{A} \mid \mathcal{C})$	$\mathcal{A} \parallel (\mathcal{B} \wedge \mathcal{C})$	$\Leftrightarrow$	$(\mathcal{A} \parallel \mathcal{B}) \wedge (\mathcal{A} \parallel \mathcal{C})$

  - Dualization and other manipulations are based on logically valid equations; these have been studied extensively for the original Ambient Logic.

The Query Language

Q ::=	Query
from $Q \vDash \mathcal{A}$ *select* Q'	match and collect
X	matching variable
0	empty result
$\eta[Q]$	nesting of result
$Q \mid Q'$	composition of results
$f(Q)$	tree functions (for extensibility)

- *from* $Q \vDash \mathcal{A}$ *select* Q'

  All the matches of Q with $\mathcal{A}$ are computed, producing bindings for the x and X variables that are free in $\mathcal{A}$. The result expression Q' is evaluated for each (distinct!) such binding, and all the results are merged by $\mid$.

- N.B.: This general approach to building a query language Q for a logic $\mathcal{A}$, is fairly independent from the details of the logic.

Query Examples

- Joins

  Merge info about persons from two db's:

  from db1 ⊨ *person*[*name*[X^λ] | Y^λ] | **T** *select*
  from db2 ⊨ *person*[*name*[X] | Z^λ] | **T** *select*
  person[*name*[X] | Y | Z]

  $^\lambda$: binding occurrence

- Restructuring

  Rearrange publications from by-article to by-year,
  for each distinct year (i.e., for each distinct binding of X):

  from db ⊨ .*article*[.*year*[X^λ]] *select*
  publications-by-year[
  year[X] |
  from db ⊨ .*article*[*year*[X] | Z^λ] *select article*[Z]]

  Z binds all fields except *year*; this is rather unusual in QL's

Query Examples

- Recursion

Find all email (or e-mail) addresses:

$$\textit{from } db \vDash \mu\xi.\ .email[X^{\lambda}] \vee .e\text{-}mail[X^{\lambda}] \vee \exists x.\ .x[\xi]$$
$$\textit{select } email[X]$$

- Unsafe queries

To be avoided by static or dynamic detection:

$$\textit{from } db \vDash (male[X^{\lambda?}] \vee female[Y^{\lambda?}]) \mid \mathbf{T} \ \ \textit{select } X^{?} \mid Y^{?}$$

$$\textit{from } db \vDash \neg author[X^{\lambda?}] \ \ \textit{select } X^{?}$$

Reference Query Semantics

- Schemas (for rows and tables)
  - $\mathbf{V} = V_1 \ldots V_n$ is a collection of distinct variables: either label variables x or tree variables X.
- Rows (or Valuations)
  - $\rho^{\mathbf{V}}$ is a row with schema $\mathbf{V}$: it maps each variable in $\mathbf{V}$ to a value of the appropriate kind (label or tree).
  - A row can be seen as the result of a match, or as an environment in which to evaluate further matches.
- Tables (or Relations)
  - A set of rows with a common schema $\mathbf{V}$ is called a table. That is, $\mathbf{V}$ names the columns of the table.
- Query Semantics $[\![Q]\!]_{\rho^{\mathbf{V}}}$
  - Produces a tree: the result of the query Q, where $\rho^{\mathbf{V}}$ is used as an environment for the free variables of Q (included in $\mathbf{V}$).

Reference Query Semantics

$$
\begin{array}{lll}
[\![X]\!]_\rho v & \triangleq & \rho^V(X) \\
[\![\mathbf{0}]\!]_\rho v & \triangleq & \mathbf{0} \\
[\![\eta[Q]]\!]_\rho v & \triangleq & \rho^V(\eta)[[\![Q]\!]_\rho v] \qquad (\rho^V(n) \triangleq n) \\
[\![Q \mid Q']\!]_\rho v & \triangleq & [\![Q]\!]_\rho v \mid [\![Q']\!]_\rho v \\
[\![f(Q)]\!]_\rho v & \triangleq & f([\![Q]\!]_\rho v)
\end{array}
$$

$$
[\![\mathit{from}\ Q \vDash \mathcal{A}\ \mathit{select}\ Q']\!]_\rho v \triangleq
$$

$$
\bigcup{}^{+} \textbf{ for } \rho'^{V'} \supseteq \rho^V \textbf{ s.t. } V' = V \cup FV(\mathcal{A}) \wedge [\![Q]\!]_\rho v \in [\![\mathcal{A}]\!]_{\rho'} v'_{,\varnothing}
$$

$$
\textbf{of } [\![Q']\!]_{\rho'} v'
$$

- $[\![\mathit{from}\ Q \vDash \mathcal{A}\ \mathit{select}\ Q']\!]_\rho v$

  Consider all the valuations ρ' that extend the current valuation ρ with the (still-free) variables of $\mathcal{A}$, and such that Q matches $\mathcal{A}$ under the appropriate valuations.
  For all such (distinct) valuations, compute Q' and put all the results in parallel.

Just a "reference" semantics

- The reference semantics is clean, but is hopelessly inefficient.
  - Literally, it requires computing beforehand a potentially infinite set of $\rho'^{V'} \supseteq \rho^V$, which are then filtered by *checking* that $[\![Q]\!]_\rho v \in [\![\mathcal{A}]\!]_{\rho', v', \varnothing}$. The $\rho'^{V'}$ that survive (hopefully a finite set) are used to build the result.
  - It should be much better to compute this (hopefully) finite set of useful $\rho'^{V'}$ on the fly, by *matching* Q to $\mathcal{A}$.
- This idea leads to a *table algebra*, used for building the relevant valuations while matching Q to $\mathcal{A}$, and to a refined evaluation procedure.

The Table Algebra

- A relational-style algebra for *relations over labels & trees*:

$\mathbf{1}^{\mathrm{V}}$		the largest table with schema **V**			
$R^{\mathrm{V}} \cup^{\mathrm{V}} R'^{\mathrm{V}}$	$\triangleq$	$R^{\mathrm{V}} \cup R'^{\mathrm{V}}$	$\subseteq$	$\mathbf{1}^{\mathrm{V}}$	
$Co^{\mathrm{V}}(R^{\mathrm{V}})$	$\triangleq$	$\mathbf{1}^{\mathrm{V}} \setminus R^{\mathrm{V}}$	$\subseteq$	$\mathbf{1}^{\mathrm{V}}$	
$R^{\mathrm{V}} \times^{\mathrm{V,V'}} R'^{\mathrm{V'}}$	$\triangleq$	$\{\rho;\rho' \parallel \rho\in R^{\mathrm{V}}, \rho'\in R'^{\mathrm{V'}}\}$	$\subseteq$	$\mathbf{1}^{\mathrm{V}\cup\mathrm{V'}}$	$(\mathrm{V}\cap\mathrm{V'}=\emptyset)$
$\Pi^{\mathrm{V}}_{\mathrm{V'}} R^{\mathrm{V}}$	$\triangleq$	$\{\rho'\in\mathbf{1}^{\mathrm{V'}} \parallel \exists\rho\in R^{\mathrm{V}}.\ \rho\supseteq\rho'\}$	$\subseteq$	$\mathbf{1}^{\mathrm{V'}}$	$(\mathrm{V'}\subseteq\mathrm{V})$
$\sigma^{\mathrm{V}}_{\eta\sim\eta'} R^{\mathrm{V}}$	$\triangleq$	$\{\rho\in R^{\mathrm{V}} \parallel , \rho^{\mathrm{V}}(\eta) \sim \rho^{\mathrm{V}}(\eta')\}$	$\subseteq$	$\mathbf{1}^{\mathrm{V}}$	$(FV(\eta,\eta')\subseteq\mathrm{V})$

- Derived operators

$Ext^{\mathrm{V}}_{\mathrm{V'}}(R^{\mathrm{V}})$	$\triangleq$	$R^{\mathrm{V}} \times^{\mathrm{V,V'\setminus V}} \mathbf{1}^{\mathrm{V'\setminus V}}$	$\subseteq \mathbf{1}^{\mathrm{V'}}$	$(\mathrm{V}\subseteq\mathrm{V'})$
$R^{\mathrm{V}} \cap^{\mathrm{V}} R'^{\mathrm{V}}$	$\triangleq$	$Co^{\mathrm{V}}(Co^{\mathrm{V}}(R^{\mathrm{V}}) \cup^{\mathrm{V}} Co^{\mathrm{V}}(R'^{\mathrm{V}}))$	$\subseteq \mathbf{1}^{\mathrm{V}}$	
$R^{\mathrm{V}} \bowtie^{\mathrm{V}\cup\mathrm{V'}} R'^{\mathrm{V'}}$	$\triangleq$	$Ext^{\mathrm{V}}_{\mathrm{V}\cup\mathrm{V'}}(R^{\mathrm{V}}) \cap^{\mathrm{V}\cup\mathrm{V'}} Ext^{\mathrm{V'}}_{\mathrm{V}\cup\mathrm{V'}}(R'^{\mathrm{V'}})$	$\subseteq \mathbf{1}^{\mathrm{V}\cup\mathrm{V'}}$	
…				

Query Evaluation

- We define a refined query semantics:
  - A procedure $\mathsf{Q}(Q)_\rho\mathrm{v}$ that evaluate queries, uses a procedure $\mathsf{B}(I, \mathcal{A})_\rho\mathrm{v}$ to evaluate binders $Q \vDash \mathcal{A}$ in the *from-select* case.
  - $\mathsf{B}(I, \mathcal{A})_\rho\mathrm{v}$ produces a table, and is computed using the table algebra operators.
  - Natural join, $\bowtie$, takes the role of "unification" for match-variables in binders.
- Some cases (simplified for non-recursive formulas):

$$\mathsf{Q}(\mathit{from}\ Q \vDash \mathcal{A}\ \mathit{select}\ Q')_\rho\mathrm{v} \quad \triangleq$$

$$\mathit{let}\ I = \mathsf{Q}(Q)_\rho\mathrm{v}\ \mathit{and}\ R^{FV(\mathcal{A})\backslash\mathrm{V}} = \mathsf{B}(I, \mathcal{A})_\rho\mathrm{v}\ \mathit{in}\ \bigcup^{+}{}_{\rho'\in R}\ \mathsf{Q}(Q')_{(\rho\mathrm{v};\rho')}$$

$$\mathsf{B}(I, \neg\mathcal{A})_\rho\mathrm{v} \quad \triangleq \quad Co^{FV(\mathcal{A})\backslash\mathrm{V}}(\mathsf{B}(I, \mathcal{A})_\rho\mathrm{v})$$

$$\mathsf{B}(I, \mathcal{A} \wedge \mathcal{B})_\rho\mathrm{v} \quad \triangleq \quad \mathsf{B}(I, \mathcal{A})_\rho\mathrm{v} \bowtie^{FV(\mathcal{A})\backslash\mathrm{V}, FV(\mathcal{B})\backslash\mathrm{V}} \mathsf{B}(I, \mathcal{B})_\rho\mathrm{v}$$

$$\mathsf{B}(I, \exists X.\mathcal{A})_\rho\mathrm{v} \quad \triangleq \quad \Pi^{FV(\mathcal{A})\backslash\mathrm{V}}{}_{FV(\mathcal{A})\backslash\mathrm{V}\backslash\{X\}}\ \mathsf{B}(I, \mathcal{A})_\rho\mathrm{v}$$

Correctness Theorem

- I.e., instead of computing the set $[\![\mathscr{A}]\!]$, as in the reference semantics, and checking at the end whether I is in $[\![\mathscr{A}]\!]$, we process I and $\mathscr{A}$ together in B, step by step.
- The table algebra is still specified rather abstractly. Any particular implementation of the table algebra yields an implementation of B, and hence of the query procedure Q.
- Correctness: the query evaluation procedure conforms to the reference semantics, so it is correct:

$$\forall Q.\ \forall \mathbf{V} \supseteq FV(Q).\ \forall \rho^{\mathbf{V}}. \qquad \mathsf{Q}(Q)_\rho \mathrm{v}\ =\ [\![Q]\!]_\rho \mathrm{v}$$

By a simple induction, with a non-trivial lemma in the recursive formulas case.

What's left out

- Effective implementations of the table algebra:
  - In general, the tables manipulated by B can get infinite, e.g. for unsafe queries, or for negation (may be ok if final result is finite).
  - To supply a real implementation, one must provide a *concrete (sub-)algebra* that provides a particular, efficient, representation of tables, and of the operators over them. Some techniques:
    - Eliminate unsafe queries by static or dynamic checks.
    - Implement important derived operators directly (typically $\bowtie$, which behaves finitely over finite tables).
    - Push negation to the leaves (define B over dualized logical operators, map them to the algebra).
    - Represent certain infinite tables by finite means (e.g. by constraints) and define the operators to work over those.
  - These techniques are being investigated in an implementation (**TQL**) that Giorgio Ghelli is carrying out in Pisa.

Conclusions

- There are many proposals for SSD query languages. Given the power of recursive formulas, we think we can capture many of them (certainly not all), and at least identify a natural spot in design space.
- We have investigated the notion of *SSD query algebra*, which has been missing for too long. (Other proposals are now emerging.)
- We have provided a query language for SSD, a set of logical optimization/rewrite rules, a reference semantics, a query algebra, a specification of algebra-based implementations, and a correctness theorem for the specification w.r.t. the reference semantics.

- L.Cardelli, G.Ghelli: ***A query language based on the ambient logic***. Proc. ESOP'01 (invited paper). <http://www.luca.demon.co.uk>
- G.Ghelli: ***Evaluation of TQL queries.*** To appear. <http://www.di.unipi.it/~ghelli/papers.html>