From complex-object to semistructured query languages
Complex objects

Types:

\[ \tau \rightarrow 0 \]
| \([\tau, \ldots, \tau]\) |
| \{\tau\} |

Values of type 0 are atomic data elements

Values of type \([\tau_1, \ldots, \tau_n]\) are tuples \([v_1, \ldots, v_n]\) with \(v_i\) a value of type \(\tau_i\)

Values of type \{\tau\} are finite sets of values of type \(\tau\)

- Relational algebra knows only “flat” types: of the form \{[0, \ldots, 0]\}
Operations on complex objects

Tuples: projection, tuple formation

Sets: union, singleton formation

⇒ Build a programming language around these operations by adding

• if-then-else

• structural recursion
Structural recursion

Any function $f$ of type $\tau \rightarrow \{\sigma\}$ yields a function $\bar{f}$ of type $\{\tau\} \rightarrow \{\sigma\}$ by structural recursion:

$$\bar{f}(\emptyset) := \emptyset$$
$$\bar{f}(\{x\}) := \{f(x)\}$$
$$\bar{f}(s_1 \cup s_2) := \bar{f}(s_1) \cup \bar{f}(s_2)$$

Equivalently:

$$\bar{f}(s) := \bigcup \{f(x) \mid x \in s\}$$

[Backus; Bird; Meertens]
The nested relational calculus (NRC)

[Buneman, Tannen, Wong]

Typed variables $x^\tau : \tau$

\[
\frac{e_1, e_2 : \sigma \quad e_3, e_4 : \tau}{\text{if } e_1 = e_2 \text{ then } e_3 \text{ else } e_4 : \tau}
\]

Tuples:

\[
\frac{e : [\tau_1, \ldots, \tau_n]}{\pi_i(e) : \tau_i} \quad (i = 1, \ldots, n)
\]

\[
\frac{e_1 : \tau_1 \quad \ldots \quad e_n : \tau_n}{[e_1, \ldots, e_n] : [\tau_1, \ldots, \tau_n]}
\]
Sets:

\[
\begin{array}{c}
\emptyset \tau \colon \{\tau\} \\
\{e\} \colon \{\tau\} \\
\end{array}
\quad
\begin{array}{c}
e \colon \tau \\
\end{array}
\quad
\begin{array}{c}
e_1 : \{\tau\} \\
\end{array}
\quad
\begin{array}{c}
e_2 : \{\tau\} \\
\end{array}
\quad
\begin{array}{c}
e_1 \cup e_2 : \{\tau\} \\
\end{array}
\]

Structural recursion:

\[
\begin{array}{c}
e_1 : \{\sigma\} \\
\end{array}
\quad
\begin{array}{c}
e_2 : \{\tau\} \\
\end{array}
\quad
\begin{array}{c}
\bigcup\{e_1 \mid x \in e_2\} : \{\sigma\} \\
\end{array}
\]

(x becomes bound)

An expression \( e \colon \tau \)

with free variables \( x_1^{\tau_1}, \ldots, x_k^{\tau_k} \)

expresses a function of type

\[
\tau_1 \times \cdots \times \tau_n \rightarrow \tau
\]
Example

\[ f : \{\{0\}\} \times \{\{0\}\} \rightarrow \left\{ \{\{0\}, \{0\}, \{0\}\} \right\} \]
\[ x, y \mapsto \{ [u, v, u \cap v] \mid u \in x \& v \in y \} \]
\[ \bigcup \bigcup \{ [u, v, u \cap v] \mid u \in x \} \mid v \in y \} \]
\[ \bigcup \{ \text{if } z \in v \text{ then } \{z\} \text{ else } \emptyset \mid z \in u \} \]
\[ \bigcup \{ \text{if } z' = z \text{ then } \{z\} \text{ else } \emptyset \mid z' \in v \} = \{z\} \]
NRC is the “right” extension of FO to complex objects

In particular, the expressions of type

\[ \tau_1 \times \cdots \times \tau_k \rightarrow \tau \]

where

1. \( \tau_1, \ldots, \tau_k, \tau \) are flat

2. types of all bound variables are also flat

correspond exactly to FO.
From complex objects to semistructured data

Strict typing implies limitations on data structures:

- no heterogeneous sets

- fixed bound on height

⇒ Arbitrary hereditarily finite sets with urelements:

- \( \emptyset \in \text{HF}(U) \)

- if \( a_1, \ldots, a_m \in U \) and \( s_1, \ldots, s_n \in \text{HF}(U) \) then also \( \{a_1, \ldots, a_m, s_1, \ldots, s_n\} \in \text{HF}(U) \)
Going all the way: untyped NRC

- Untyped variables

- if $e_1 = e_2$ then $e_3$ else $e_4$

- $\{e\}, e_1 \cup e_2$

- $\{e_1 \mid x \in e_2\}$

“Rudimentary” or “basic” set-theoretic operations [Jensen; Gandy]

Basis for suite of “$\Delta$-languages” [Sazonov, Lisitsa]
An intermediate: semistructured query languages

Two sorts of variables: atomic ("label") and set ("tree")

Allow equality test on atomic variables only

⇒ Satisfiability becomes decidable when $\mathcal{U}$ is finite

"Surface syntax" of UnQL [Buneman, Fernandez, Suciu]
Vertical and horizontal transitive closure

We can still dive only until a fixed depth inside the data structures ⇒ add recursion

Basic, “vertical” TC operator is very typical:

$$TC\left(\left\{\left\{a\right\}\right\}\right) = \left\{\left\{a\right\}, \{a\}, a\right\}$$

For more power:

- “Horizontal” TC operator, as in FO(TC), in $\Delta$-languages

- In StruQL the same is achieved by composing queries

- Alternatively, UnQL proposes a more powerful form of structural recursion on trees (and even graphs), but this becomes very messy
Bounded-height creation

Output is always a set constructed from sets in the TC of input.

Not counting heights of these sets, height of output is bounded by a constant fixed by the query.

⇒ Cannot express transformation:

\[
\{[a_1, a_2], [a_2, a_3], \ldots, [a_{n-1}, a_n], [a_n, b]\}
\]

\[\mapsto \{\cdots \{b\} \cdots\} \text{ (height } n\text{)}\]

\(\Delta\)-languages provide “Mostowski collapsing” operator for going from the \(\in\)-graph of a set to the set itself
Stepping back

A HF set over $U$ is nothing but a tree with two kinds of nodes: sets ("objects") and elements of $U$ ("values")

No reason not to generalize this to arbitrary two-sorted structures

Mappings among such structures can then be expressed using interpretations in, say, FO(TC)

- Need notion of interpretation where input values retain their identity in the output

$\Rightarrow$ Refine basic isomorphism to $U$-isomorphism

StruQL [Fernandez, Florescu, Levy, Suciu]

OO query languages! [e.g., GOOD]