Querying Hyperset/Web-Like Databases

Vladimir Sazonov

University of Liverpool

V.Sazonov@csc.liv.ac.uk

http://www.csc.liv.ac.uk/~sazonov

The **goal** of this talk is in fact modest:

- to give a simple presentation of *hyperset approach* to semistructured (web-like) databases and of the corresponding *query language* Δ,

- to answer the question on how *expressive power* of querying to semistructured databases can be characterised

  – in terms of *descriptive complexity*,

  – by defining *formal semantics of path expressions* in terms of Δ,

  – by defining *formal semantics of other known languages* UnQL and UnCAL to semistructured data (based also on *structural recursion*) which are *most close to* Δ.

Expressive Power?

Some citations:

[S. Abiteboul, P. Buneman, D. Suciu, Data on the Web: From Relations to Semistructured Data and XML, 2000]:

"There are simple restructuring problems that defeat all the query languages we shall describe. Moreover, there is **no accepted notion of completeness for semi-structured data restructuring***. Therefore, it is yet* **unclear what expressive power** *should capture a query language for semistructured data."*

[L. Cardelli, Semistructured Computation, 2000]:

"It should be noted that **basic questions of expressive power** *for semistructured database query languages are* **still open***."*

Expressive Power?

Hyperset approach resolves this problem for semistructured databases represented as *finite graphs with labelled edges* where *no ordering* is assumed on the nodes.

— **Not** the case of XML where the data are ordered, although the ideas can evidently be applied to.

It consists in using the concept of

- *(finite) set* of sets of sets, etc., even *allowing cycles*, like in the simplest proper hyperset

  $$\Omega = \{\Omega\}$$

  in its full generality as a straightforward and transparent **abstract model of arbitrary data** *having no predefined structure*.

- Abstract *hyperset theory* with some applications is described in books **[Aczel 1988]** and **[Barwise, Moss 1996]**.

- This is also a very natural way of *generalising the ordinary relational view to the case of semistructured databases*.

Relational Databases as a Restricted Set-theoretic Approach to DB

Relation Name			
Attr_1	Attr_2	...	Attr_N
value_1	value_2	...	value_N
value'_1	value'_2	...	value'_N
$\vdots$	$\vdots$	$\ddots$	$\vdots$

STUDENTS			
NAME	BIRTH-DATE	...	DEPT
I. Ivanov	1981	...	CS
A. Petrov	1979	...	Math
P. Sidorov	1980	...	Physics

Database state is a finite *set* of relations.

$$\mathsf{DB} = \{\mathsf{R1}, \mathsf{R2}, \ldots, \mathsf{RN}\}$$

Relation is a finite *set* of tuples (rows, records).

$$\mathsf{STUDENTS} = \{\mathsf{student1}, \mathsf{student2}, \mathsf{student3}\}$$

Tuple is also a finite *set* of *labelled "atomic" values*:

$$\mathsf{student1} = \{\mathsf{NAME} : \mathrm{I.Ivanov}, \mathsf{BIRTH\text{-}DATE} : 1981, \mathsf{DEPT} : \mathsf{CS}\}$$

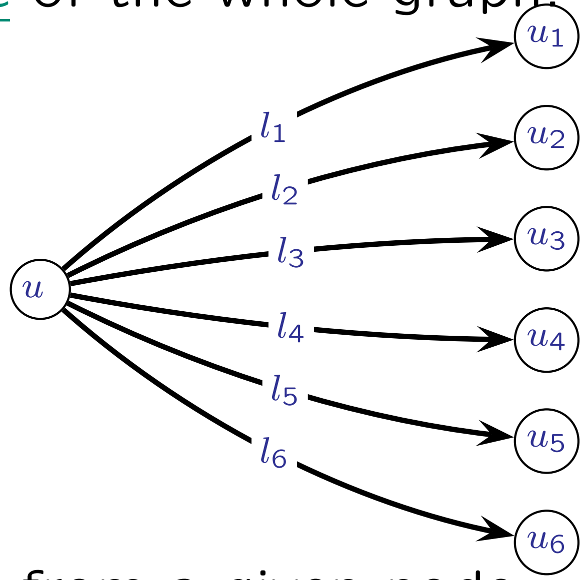
Therefore relational DB is a *set of sets of sets* of $\mathit{Nesting} = 3$.

- Why not *arbitrary nesting*? (Nested relational databases).
- What about *cycling* (and, in general, *Web-like Databases*)?
  - CS may contain further "deep" information, say referring via a chain of "clicks" again to the STUDENTS table.
- Thus, we need also "cycling" sets, like $\Omega = \{\Omega\}$ which exist in *non-well-founded set theory* and are also called *hypersets*.

Semistructured Databases (SSD): Graphs or Hypersets?

Navigational view on SSD/WWW/WDB as a graph

Fork or fan fragment of the whole graph:



Only edges *outgoing* from a given node u are shown.

Corresponds to "mouse clicks" on the labels l_i on the Web page with **URL** u.

The same set theoretically:

$u = \{l_1 : u_1, \ldots, l_n : u_n\}$

Thus, graph nodes *represent* (hyper)*sets* consisting of *labelled elements*.

Semistructured Databases (SSD): Graphs or Hypersets?

Navigational view on SSD/WWW/WDB *leads to*

- *graph representation* of data, and also to
- stressing on *path expressions* as the basic part of corresponding query languages such as Lorel or UnQL or XPath (the latter devoted to XML).

Hyperset view is

- more *abstract*, but
- completely *consistent with navigational view*, and
- *bears ideas from set theory* rather than from graphs.

Hypersets and *graphs* are **inseparable**,
and serve as *two levels of abstractions* in hyperset approach:

- *conceptual, or denotational level* (hypersets), and
- *computational level* (graphs)

Hypersets represented by set equations

Consider any (recursive) *system of set equations* like

$$s_5 = \{l_1 : s_3, l_2 : s_5, l_2 : s_3\}.$$

In general,

$$\text{WDB} : \begin{cases} s_1 & = & B_1(s_1, \dots, s_n), \\ s_2 & = & B_2(s_1, \dots, s_n), \\ & \dots & \\ s_n & = & B_n(s_1, \dots, s_n). \end{cases},$$

or in *vector form*:

$$\text{WDB} : \{\bar{s} = \bar{B}(\bar{s})\}.$$

This is actually a *textual representation of an arbitrary finite directed graph* with labelled edges where *set names* s_i serve as *nodes*.

Elements of the set $s_5 = \{l_1 : s_3, l_2 : s_5, l_2 : s_3\}$ represent *outgoing from* s_5 *edges* labelled accordingly:

$$s_5 \xrightarrow{l_1} s_3, \quad s_5 \xrightarrow{l_2} s_5, \quad s_5 \xrightarrow{l_2} s_3$$

Hypersets represented by set equations

Antifoundation Axiom (AFA) of hyperset theory says that each such system of equations has a *unique solution* in the abstract universe of hypersets.

This way we achieve a *natural level of abstraction* making the hyperset approach to SSD/WDB very attractive.

In particular, two set names s_i and s_j could denote the *same* abstract hyperset,

$$s_i = s_j.$$

This means that in our database WDB s_i and s_j are *informationally indistinguishable*, or *bisimular*.

Otherwise, they are called *informationally distinguishable*, or *non-bisimular*, $s_i \neq s_j$.

Hypersets represented by set equations

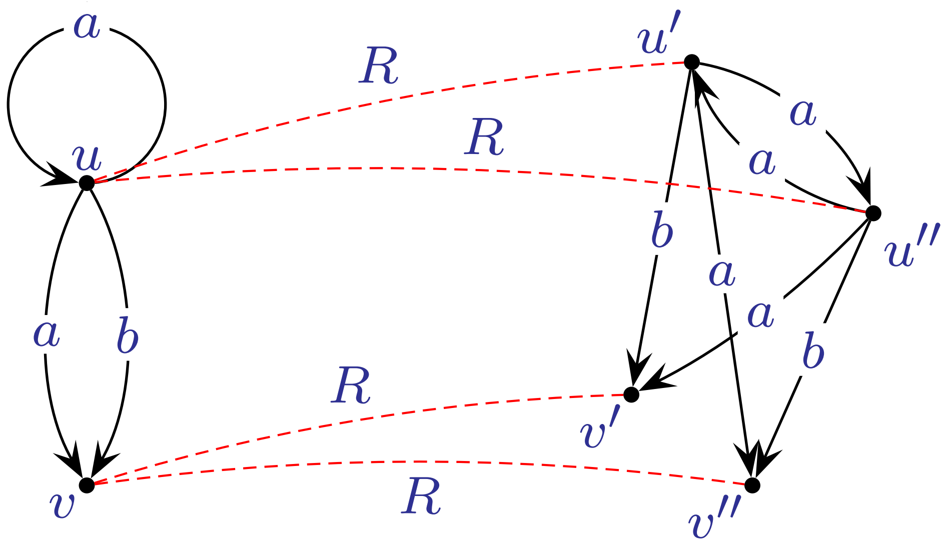
Distinguishability of two sets can be checked by the repeated applying the following two symmetric *derivation rules* (for simplicity without labels)

$$s_i \neq s_j \Leftarrow \exists w \in s_i \forall v \in s_j (w \neq v)$$
$$s_j \neq s_i \Leftarrow \exists w \in s_i \forall v \in s_j (w \neq v)$$

meaning that two sets s_i and s_j are different (distinguishable) **if** they *have different elements*.

Hypersets represented by set equations

Knowing equality (bisimulation) relation allows for *removing redundancies* in the data:




```
u = {a:u,a:v,b:v}
v = {}
```

```
u1 = {a:u2,b:v1,a:v2}
u2 = {a:u1,a:v1,b:v2}
v1 = {}
v2 = {}
```

Distributed WDB

WDB, as a *system of set equations*, can be

- *distributed* between several files,
- even on various *remote sites*.

These can be, for example, HTML files or XML files of special form which should be *hyperlinked*.

If some set name is described in another file the link should lead to the appropriate set equation in that file.

Hyperset Query Language Δ

the basic [Gandy 74] or rudimentary [Jensen 72] fragment

Define inductively Δ-*formulas* and Δ-*terms* by the clauses

$$\langle\Delta\text{-term}\rangle \quad ::= \quad \langle\text{set variable or constant}\rangle \,\Big|\, \emptyset \,\Big|\, \{l_1 : a_1, \ldots, l_n, a_n\} \,\Big|\, \bigcup a \,\Big|\, \mathrm{TC}(a) \,\Big|\, \{l : t(x,l) \mid l : x \in a \;\&\; \varphi(x,l)\}$$

$$\langle\Delta\text{-formula}\rangle \quad ::= \quad a = b \,\Big|\, l_1\, R\, l_2 \,\Big|\, l_0 : a \in b \,\Big|\, \varphi \;\&\; \psi \,\Big|\, \varphi \vee \psi \,\Big|\, \neg\varphi \,\Big|\, \forall l : x \in a.\varphi(x,l) \,\Big|\, \exists l : x \in a.\varphi(x,l)$$

The *intuitive meaning* is just well-known set theoretical one.

Note that only *bounded quantifiers* are allowed.

Since all (hyper)sets are assumed to be *finite*, this makes the values of Δ-terms and Δ-formulas computable, and even *in polynomial time*.

Moreover, a natural extension of the above syntax by a *recursion* (**Rec**) and hyperset *decoration* (**Dec**) operators captures *exactly* PTIME over (finite) *hypersets* **[Sazonov 1987,93,94]**, **[Lisitsa,Sazonov 1997]**.

Another version of Δ captures *exactly* LOGSPACE over (finite) *well-founded sets* **[Lisitsa,Sazonov 1997]**, **[Leontjev,Sazonov 2001]**.

Recursion operator

Recursive Δ-Separation is straightforward extension from the framework of first order logic to Δ:

the following is considered as new Δ-*term* construct

$$\textbf{Rec}\ p.[p = \{l : x \in a \mid \varphi(l, x, p)\}]$$

for φ any Δ-formula (depending on set variable p *positively*).

Let $p_0 \stackrel{\text{def}}{=} \emptyset$ and $p_{i+1} \stackrel{\text{def}}{=} \{l : x \in a \mid \varphi(l, x, p_i)\}$.

Then this *monotonic* sequence of subsets of a must *stabilise*, and the result is considered as the *value of the recursion operator*.

Decoration operator

For *any* hyperset g, some its elements may happen to be *ordered pairs*

$$\langle u, v\rangle \stackrel{\text{def}}{=} \{\mathsf{Fst} : u, \mathsf{Snd} : v\}.$$

In general, these are *labelled ordered pairs* $l : \langle u, v\rangle$ in g considered also as *labelled edges* $u \xrightarrow{l} v$ *of a graph* represented in this way by the hyperset g.

All other *non-pair elements* of g may be *ignored*.

Thus,

- *any (hyper)set* g can be considered as a *graph*,
- and *any other set* v can be considered as its *vertex* (possibly isolated, if v does not participate in the above ordered pairs).

Decoration operator

$$\mathbf{Dec}(g,v) = \mathbf{Dec}_g(v)$$

is the *unique hyperset* represented by the vertex v of g according to **AFA** for graphs.

Intuitively, **Dec** is also a *plan performance operator:*

v-rooted graph (g,v) serves as a *plan of building of a new hyperset* (or complex data) $\mathbf{Dec}(g,v)$, according to this plan.

Decoration operator

Example 1.

$$g = \{\langle u, v\rangle, \langle v, w\rangle, \langle u, w\rangle\}$$

represents the (acyclic) graph

$$u \longrightarrow v, \quad v \longrightarrow w, \quad u \longrightarrow w.$$

Then $\mathbf{Dec}_g(w) = \emptyset$, $\mathbf{Dec}_g(v) = \{\emptyset\}$, and $\mathbf{Dec}_g(u) = \{\{\emptyset\}, \emptyset\}$,

Example 2. For the *loop-graph*  $= \{v \to v\}$,

$$\mathbf{Dec}_{\circlearrowleft}(v) = \Omega$$

is (according to **AFA**) a *unique hyperset* which satisfies, the equality

$$\Omega = \{\Omega\}.$$

This "cyclic" hyperset *is not definable* in Δ without $\mathbf{Dec}$.

Decoration operator

These examples are very simple, but rather abstract.

In more practical flavour:

Graph g can describe a *plan* of constructing a system of Web or WDB pages.

Then the action of **Dec** would be something like a *transformation of this plan to a real system of* WDB *pages hyperlinked as required in this plan*.

Thus, **Dec** is a quite reasonable WDB construct.

Operational semantics of Δ-queries

Given any WDB as a system of set equations $\bar{s} = \bar{B}(\bar{s})$, we want to compute

- *hyperset value* of any Δ-term t or
- *Boolean value* of any Δ-formula φ,

assuming that they *involve only set names* (constants) $\bar{s}$ from this WDB and *no free variables*.

The process of *computation* consists in a sequence of *reduction steps*

$$\text{WDB}_0 \rhd \text{WDB}_1 \rhd \text{WDB}_2 \rhd \ldots \rhd \text{WDB}_N = \text{WDB}'.$$

WDB_0 is the initial WDB **extended by <u>non-flat</u> equation** $res = t$.

The last $\text{WDB}' = \text{WDB}_N$ consists only of flat set equations, including the result of "**flattening**" the equation $res = t$.

The unique *abstract hyperset* denoted by res in $\text{WDB}' = \text{WDB}_N$ is considered as the **computed value** of t.

In the case if the query is a Δ-formula φ, we include in WDB_0 the *Boolean equation* $res = \varphi$ with the intention to get a "*flattened*" value **true** or **false**.

Operational semantics of Δ-queries
(the pure case)

Let us assume that $s, p, \ldots, q$, participating in the following below reduction rules, be some *selected* set names with corresponding (recursive) set equations from the initial WDB:

$$\begin{array}{lcl} s & = & \{s_1, \ldots, s_m\}, \\ p & = & \{p_1, \ldots, p_n\}, \\ & \ldots & \\ q & = & \{q_1, \ldots, q_k\}. \end{array}$$

Names $s_i, p_j, \ldots, q_l$ *may coincide* one with another, or with some of $s, p, \ldots, q$.

Reduction Rules:

Here res is just the left-hand-side of the equation to be reduced.

$$res = t(t_1, \ldots, t_a) \rhd \left\{ \begin{array}{lcl} res & = & t(res_1, \ldots, res_a), \\ res_1 & = & t_1, \\ & \ldots & \\ res_a & = & t_a, \end{array} \right.$$

$res = \{s, p, \ldots, q\}$ — no reduction is required for set names $s, p, \ldots, q$

$res = s \cup p \cup \ldots \cup q \rhd res = \{s_1, \ldots, s_m, p_1, \ldots, p_n, \ldots, q_1, \ldots, q_k\}$,

Reduction Rules

$res = \bigcup s \rhd res = s_1 \cup \ldots \cup s_m,$

$res = \{t(x) \mid x \in p\} \rhd res = \{t(p_1), \ldots, t(p_n)\},$

$res = \{x \in p \mid \varphi(x)\} \rhd res = \{p_{i_1}, \ldots, p_{i_{n'}}\}$ with p_{i_j} all those set names p_i

for which $res_i = \varphi(p_i) \rhd res_i =$ **true**

(and $res_i = \varphi(p_i) \rhd res_i =$ **false** for all other i),

$res = \forall x \in p\, \varphi(x) \rhd res = \varphi(p_1)\ \&\ \ldots\ \&\ \varphi(p_n)$ ($=$ **true** if $n = 0$),

$res =$ **true** & **true** $\rhd res =$ **true**,

$res = \varphi$ & **false** $\rhd res =$ **false**,

$res =$ **false** & $\varphi \rhd res =$ **false**,

$res = \neg$**true** $\rhd res =$ **false**,

$res = \neg$**false** $\rhd res =$ **true**,

$res = s \in p \rhd res = \exists x \in p\, (s = p),$

$res = (s = p) \rhd \ldots$ (bisimulation relation has been considered above).

We **omit** the rules for *transitive closure*, *recursion*, and *decoration*, operators TC, **Rec**, and **Dec**.

Queries preserve equality

All Δ-queries over hypersets definable in terms of systems of set equations and the above operational semantivs are in *complete accordance* with the bisimulation equivalence relation:

- they are *bisimulation invariant*
- that is, *preserve the equality relation between abstract hypersets*:

$$\bar{x} = \bar{y} \implies q(\bar{x}) = q(\bar{y}), \quad \text{if } q \text{ is a } \Delta\text{-term,}$$
$$\bar{x} = \bar{y} \implies q(\bar{x}) \Leftrightarrow q(\bar{y}), \quad \text{if } q \text{ is a } \Delta\text{-formula}$$

where $\bar{x}$ and $\bar{y}$ are set names. For **example**,

$$s_i = s_j \implies \bigcup s_i = \bigcup s_j.$$

This also means that $=$ is a *congruence* relation.

It is by this reason that we denote bisimulation just as $=$.

This represents more **abstract set theoretic level of thought** —**not** a graph theoretic one—although closely related.

Path expressions

In general, define *path expression* either as

- Λ, the *empty path expression*, or
- a *label constant*, or
- a *label variable*, or
- $\_$, *underscore* (meanig "any label"), or
- $p.q$, *concatenation* of path expressions, or
- $q\langle z\rangle p$, *concatenation by a set variable or a set name* z, or
- $(p)^*$, *iteration*, or
- $(p)^?$, *optional* path expression, or
- $p|q$, *alternation* of path expressions (either p or q), or
- $p||q$, *parallel composition or branching* (both p and q).

Path expressions defined above extend those in UnQL called there *patterns*.

Path expressions

Before including path expressions in Δ, let us give formal inductive definitions of two *auxiliary kinds of expressions*

- *the assertion* $p\langle t\rangle$: "there exists, possibly $||$-*branching* path p from t via all vertices and labels mentioned in p", and

- *the set* $[p\langle t\rangle] \subseteq \mathsf{TC}(\{t\})$: "the set of all x such that there exists a path p from t via all vertices and labels mentioned in p all whose branches *converge* to the same x (denoted also as $\langle x\rangle p\langle t\rangle$)".

In fact, $p\langle t\rangle$ and $[p\langle t\rangle]$ are a Δ-*formula* and Δ-*term*, respectively, depending on set and label variables occurring in p.

The above should be understood over any *non-redundand (strongly extensional)* WDB-graph.

Defining $p\langle t\rangle$ and $[p\langle t\rangle]$ in Δ

$\langle t\rangle \stackrel{\text{def}}{=} \textbf{true}$,

$\_\langle t\rangle \stackrel{\text{def}}{=} t \neq \emptyset$,

$l\langle t\rangle \stackrel{\text{def}}{=} \exists m : x \in t\ (m = l)$,

$q.p\langle t\rangle \stackrel{\text{def}}{=} \exists y \in [p\langle t\rangle](q\langle y\rangle)$,

$q\langle z\rangle p\langle t\rangle \stackrel{\text{def}}{=} z \in [p\langle t\rangle]\ \&\ q\langle z\rangle$,

$(q)^*\langle t\rangle \stackrel{\text{def}}{=} \textbf{true}$,

$(p)^?\langle t\rangle \stackrel{\text{def}}{=} \textbf{true}$,

$(p|q|\cdots|r)\langle t\rangle \stackrel{\text{def}}{=} p\langle t\rangle \vee q\langle t\rangle \vee \cdots \vee r\langle t\rangle$,

$(p||q||\cdots||r)\langle t\rangle \stackrel{\text{def}}{=} p\langle t\rangle\ \&\ q\langle t\rangle\ \&\ \cdots\ \&\ r\langle t\rangle$;

Defining $p\langle t\rangle$ and $[p\langle t\rangle]$ in Δ

$$[\langle t\rangle] \stackrel{\text{def}}{=} \{t\},$$

$$[\_\langle t\rangle] \stackrel{\text{def}}{=} \{x \mid m : x \in t\},$$

$$[l\langle t\rangle] \stackrel{\text{def}}{=} \{x \mid m : x \in t \;\&\; m = l\},$$

$$[q.p\langle t\rangle] \stackrel{\text{def}}{=} \bigcup\{[q\langle y\rangle] \mid y \in [p\langle t\rangle]\},$$

$$[q\langle z\rangle p\langle t\rangle] \stackrel{\text{def}}{=} [q\langle z\rangle] \textbf{ if } z \in [p\langle t\rangle],$$

$$\begin{aligned}[(q)^*\langle t\rangle] &\stackrel{\text{Rec}}{=} \{t\} \cup \{x \mid \exists y \in [q\langle t\rangle]\,(x \in [(q)^*\langle y\rangle])\}\\ &\stackrel{\text{Rec}}{=} \{x \mid x = t \vee \exists y \in [q\langle t\rangle]\,(x \in [(q)^*\langle y\rangle])\}\\ &\stackrel{\text{Rec}}{=} \{x \in \mathsf{TC}(\{t\}) \mid x = t \vee \exists y \in [q\langle t\rangle]\,(x \in [(q)^*\langle y\rangle])\},\end{aligned}$$

$$[(p)^?\langle t\rangle] \stackrel{\text{def}}{=} [p\langle t\rangle] \cup \{t\},$$

$$[(p|q|\cdots|r)\langle t\rangle] \stackrel{\text{def}}{=} [p\langle t\rangle] \cup [q\langle t\rangle] \cup \cdots \cup [r\langle t\rangle],$$

$$[(p||q||\cdots||r)\langle t\rangle] \stackrel{\text{def}}{=} [p\langle t\rangle] \cap [q\langle t\rangle] \cap \cdots \cap [r\langle t\rangle].$$

Defining $p\langle t\rangle$ and $[p\langle t\rangle]$ in Δ

In particular,

$$\begin{aligned}
&\langle x\rangle l\langle t\rangle \iff l : x \in t,\\
&\langle z\rangle p\langle t\rangle \iff z \in [p\langle t\rangle],\\
&[\langle z\rangle p\langle t\rangle] = \{z\} \textbf{ if } z \in [p\langle t\rangle],\\
&q\langle z\rangle p\langle t\rangle \iff q\langle z\rangle \;\&\; \langle z\rangle p\langle t\rangle,\\
&p(\bar{z})\langle t\rangle \Rightarrow \bar{z} \in \mathsf{TC}(\{t\}).
\end{aligned}$$

The last implication is the formal confirmation of the idea that path expressions are a special case of the *transitive closure operator* TC.

This finishes the definition of semantics of the expressions $p\langle t\rangle$ and $[p\langle t\rangle]$.

Defining semantics of Δ-lanfuage extended by path expressions

Now, we extend Δ, without changing its expressive power, by new kind of terms and formulas *involving arbitrary path expressions*.

In particular, we *generalise bounded quantification*.

This extension is only *syntactic sugaring* of Δ because the new meaning is definable in the original version of the language:

$$\exists p(\bar{z})\langle t\rangle\, \varphi(\bar{z}) \overset{\text{def}}{=} \exists \bar{z} \in \mathsf{TC}(\{t\})(p(\bar{z})\langle t\rangle \;\&\; \varphi(\bar{z})),$$

$$\{s(\bar{z}) \mid p(\bar{z})\langle t\rangle \mid \varphi(\bar{z})\} \overset{\text{def}}{=} \{s(\bar{z}) \mid \bar{z} \in \mathsf{TC}(\{t\}) \mid p(\bar{z})\langle t\rangle \;\&\; \varphi(\bar{z})\},$$

$$\bigcup_{p(\bar{z})\langle t\rangle} S(\bar{z}) \overset{\text{def}}{=} \bigcup\{S(\bar{z}) \mid p(\bar{z})\langle t\rangle\}.$$

All (set and label) variables $\bar{z}$ in a path expression $p(\bar{z})$ are quantified above.

Conceptual comparison of Δ with UnQL and UnCAL

Query languages UnQL and UnCAL where introduced and developed in **[Buneman, Davidson, Hillebrand, and Suciu, 1996]** and **[Buneman, Fernandez, and Suciu, 2000]**:

— **most** *analogous* to Δ with considering a *database as a graph with labelled edges,* **up to bisimulation relation,**

— however, because of "input" and "output" vertices, the data, considered in UnQL and UnCAL are *not exactly hypersets*, although can be *imitated* by hypersets in Δ,

— *mostly treated as graphs* in UnQL and UnCAL, and the imitation in Δ goes, anyway, also in terms of graphs.

So, UnQL and UnCAL are *essentially graph languages* with

- more stress on the operational/computational meaning and
- no explicit mentioning hypersets as a fundamental concept.

Conceptual comparison of Δ with UnQL and UnCAL

In contrast, Δ *almost completely can be understood in terms of sets* and is very intuitive essentially by the same reason why set theory proved to be the most clear and widely accepted during the last century approach giving the foundation of mathematics.

On the other hand, the *opertational semantics* of Δ is also essentially *based on graphs* or, equivalently, on textual *systems of set equations*.

Elementary set theoretic concepts are, in fact, *widely known*.

XML *representation of (hyper)sets* or systems of set equations can be aslo used very naturally.

Technical comparison of Δ with UnQL and UnCAL

Graphs from hypersets

Each hyperset x *defines in* Δ *(and is itself defined by) a* **graph**

$$G(x) = \langle V(x), E(x) \rangle$$
$$V(x) \stackrel{\text{def}}{=} \mathsf{TC}(\{x\})$$
$$E(x) \stackrel{\text{def}}{=} \{l : \langle u, v \rangle \mid u, v \in V(x) \;\&\; l : v \in u\}$$

with $l : \langle u, v \rangle$ written also as the *graph edge* $u \xrightarrow{l} v$.

Then x is the natural *root* and serves as an *input* vertex of the graph $G(x)$, and evidently

$$\mathbf{Dec}(G(x), x) = x.$$

Technical comparison of Δ with UnQL and UnCAL

Append operation of UnCAL—simplest version

Define in Δ a simple version of the *append* operation for two hypersets

$$u@_l v \stackrel{\text{def}}{=} \mathbf{Dec}(G^l_{u,v}, u)$$

where the graph $G^l_{u,v}$ is the result of "glueing" together of three vertices $\{l : \emptyset\}, \emptyset \in V(u)$, and $v \in V(v)$ in graphs $G(u)$ and $G(v)$.

Here l serves to label an *output* in u to be connected with the natural root of v playing the role of the *input* in v.

That is, v is "*substituted*" in u in place of $\{l : \emptyset\}$.

Example, $\{k : \{l : \emptyset\}, m : z\}@_l v = \{k : v, m : z\}$.

Technical comparison of Δ with UnQL and UnCAL

Append operation of UnCAL**—more general versions** (some also going outside of UnCAL) for the case of

- appending *not only to leafs*
- *multiple inputs and outputs*
- multiple hypersets u_i (graphs $G(u_i)$ connected as a *chain* $\mathcal{G} = \{u_0 \to u_1 \to \cdots \to u_n\}$ (of indefinite finite length) of hypersets as

$$@(\mathcal{G}) \stackrel{\text{def}}{=} u_0 @ u_1 @ \cdots @ u_n$$

or even as *arbitrary graph* $\mathcal{G}$.

In each case this is *definable* in Δ, and the participation of *decoration* **Dec** *is, in fact, crucial* (as in the previous slide).

Technical comparison of Δ with UnQL and UnCAL:

Structural recursion

In the case of *well-founded sets* (with no cycles) the general set theoretic *vertical* (that is going deeply to elements of x, etc.) *structural recursion* schema is defined by

$$f(x) = \bigcup_{l:y \in x} e(l, y, f(y))$$

A similar recursion schema over sets

$$f(x) = g(\bigcup_{u \in x} f(x), x)$$

was introduced earlier by **[Jensen and Karp, 1967]**. Both are set theoretic analogues of *primitive recursion* and therefore *cannot be computable in polynomial time* in general.

Technical comparison of Δ with UnQL and UnCAL:
Structural recursion

Although it is *problematic to define* general structural recursion *for arbitrary (possibly cyclic) hypersets*, this can be done, and even in Δ, for its *restricted version* used in UnCAL:

$$f(x) = \bigcup_{l:y \in x} e(l,y) @ f(y) \tag{1}$$

The above general form of *append* @ is used here, as well as for even stronger *horisontal* version of structural recursion:

$$f(x,u) = \bigcup_{l:\langle u,v\rangle \in x} e(x,l,v) @ f(x,v) \tag{2}$$

which has in Δ the power *equivalent* to **Dec**.

This recursion also looks at the structure of x, but only at a superficial, near to the root of x "horizontal" level.

Expressive power of Δ vs. UnQL and UnCAL

By using the above and following below considerations we conclude that

The *expressive power* of Δ (which is exactly PTIME) is strictly *stronger* than UnQL and UnCAL.

Conjecture. It seems plausible that the general form of the *decoration* operations **Dec** is *undefinable* even in $\Delta - \mathbf{Dec} + \mathrm{UnCAL}$ and that the *horisontal structural recursion is strictly stronger that the vertical one*.

The precise expressive power of general queries in UnCAL and in UnQL remains *unclear*.

However, it is known that *all* UnCAL *queries can be expressed in* $\mathrm{FO} + TC$.

Note that UnQL and UnCAL *even do not include the equality* (bisimulation) predicate for the data—for the sake of efficiency of query evaluation.

However, the *equality/bisimulation* of data is, anyway, *necessary to use in the semantics of path expressions* (participating in UnQL).

As general equality is also not included in UnCAL, it is also unclear that UnQL, involving equality implicitly (in the semantics of path expressions), can be really interpreted in UnCAL, as it is assumed in the paper **[P. Buneman, M. Fernandez, and D. Suciu, 2000]**.

Expressive power of Δ vs. UnQL and UnCAL

There is, however, **Theorem 3** in the op. cit. implying that **the expressive power of** UnCAL *restricted to the ordinary* **relational databases** *(i.e., acyclic data graphs/trees of a very special form and the depth three) is* **exactly** *that of* FO, or of the relational calculus. Here even the full power of UnCAL is used allowing construction of arbitrary data graphs in the intermediate steps of query evaluation.

But for the case of Δ we should get here exactly $\mathrm{FO} + \mathrm{LFP}$, as well as *for arbitrary hypersets* (or graphs representing them). That is, Δ has a stronger power which, moreover, has a precise characterisation—$\mathrm{FO} + \mathrm{LFP}$, or PTIME, if there is a linear order on labels (or if there are no labels at all).

There are also *versions of* Δ *exactly corresponding to* FO+TC, or FO+DTC (in fact, $(N/D)LOGSPACE$), but only *in the case of acyclic sets/graphs*.

It would be very *interesting to* **extend** *these results on* LOGSPACE *to the case of* **arbitrary** *(cyclic) hypersets*.