The Parameterized Relational Model

V.E.Wolfengagen
Department of Advanced Computer Studies and Information Technologies
Institute for Contemporary Education "JurInfoR-MSU"
Moscow, Russia
e-mail: vew@jmsuice.msk.ru

Abstract

The intensional approach to development of a computational model for relational languages is applied. The feature of this approach is to give a unified theoretical ground for dynamic computations – taking into account the 'stages of knowledge'. The model covered in this paper establishes the purely functional and applicative computational environment.

1. Introduction

The version of relational calculus in this paper is based on a 'computational models', and more rigorously – on the *applicative* computational model [8]. Thus derived calculus is referred as *C-calculus*, or calculus by E.F. Codd [1], [2], [3]. Besides that for the needs of conceptual modeling and design of databases the more general intensional calculus is considered which is referred as R-calculus. This intensional calculus is rather close to the languages for frames [4]. This closeness is discussed in more details.

The way a relational calculus (C-calculus) was introduced by E.F. Codd, this was a non-procedural query language of a high level, based on a predicate calculus with the restriction for proper interpretations of its expressions by the *finite* relations.

In this paper, at first, C-calculus is accommodated for using aggregate functions. Range formulae are separated from alpha-definable formulae. Then, the class of terms, which basically includes constants and variables (by attributes) is extended to the 'applications' of aggregate functions to alpha-expressions. The range terms are determined in a rather special way to fit with C-algebra.

Proceedings of the 8th International Workshop on Computer Science and Information Technologies CSIT'2006
Karlsruhe, Germany, 2006

2. Computational Model for Relational Calculus (C-calculus)

The syntax of the expressions is as follows.

1) Let fix the countable set Vr_τ of variables $v_1, v_2, \ldots$ with type τ for any τ.

2) Set of terms Tm is as follows.

(i) Individual constants for domains V_τ

$$c \in \bigcup_\tau V_\tau(k)$$

are the terms where:
k – is the assignment, $k \in Asg$;
τ – is the attribute (type symbol).
Note that more rigorous writing would be c_τ with the explicit type τ, or, equivalently, $c : \tau$. Nevertheless whenever this is clear the type indications will be omitted.

(ii) individual variables v_τ are terms of type τ for any τ.

The definition of term should be extended the this will be done later.

3) The set of *formulae* is determined step by step. The set of *alphas* is determined by induction.

(i) If $R_i \in \langle R_1, \ldots, R_N \rangle$, where $\langle R_1, \ldots, R_N \rangle$ is *scheme*, then R_i is *atomic alpha* of degree $deg(R_i)$.

(ii) *Quantified prefixes* are the expressions as $(\exists v_\tau^i)$ or $(\forall v_\tau^j)$ where variables v_τ^i, v_τ^j are bound.

(iii) The set of *closed alphas* is determined by $\alpha_\nu(v_\tau^i) \rightleftharpoons R_j$ or $R_j \rightleftharpoons \alpha_\nu(v_\tau^i)$ where $v_\tau^i \in Vr_\tau$.

(iv) *Range formula* for variable v_τ^i is $\alpha_1(v_\tau^i) \vee \ldots \vee \alpha_m(v_\tau^i)$ where $m > 0$, $v_\tau^i \in Vr_\tau$ and all the α_ν, $\nu = 1, \ldots, m$ are closed alphas.

(v) By the definition *formulae* are as follows:

a) closed alphas $(\exists v_\tau^i)\alpha(v_\tau^i)$ or $(\forall v_\tau^i)\alpha(v_\tau^i)$;

b) expressions such as $t_1 \theta t_2$ where $t_1, t_2 \in Tm$ and θ is the binary relation one from $\{ =, \neq, <, >, \leq, \geq \}$;

c) for formulae ϕ, ψ the expressions

$$\neg\phi,\ \phi \wedge \psi,\ \phi \vee \psi$$

are the formulae;

d) whenever r is the range formula of variable v_τ^i, then $(\exists r)\phi$ or $(\forall r)\phi$ are formulae.

(vi) If $t_1, \ldots, t_n$ are terms, $r_1, \ldots, r_m$ are range formulae of $v^1, \ldots, v^m$ respectively, and variables in $t_1, \ldots, t_n$ are the same as the set $\{v^1, \ldots, v^m\}$ and ϕ is formula then

$$\langle t_1, \ldots, t_n \rangle | r_1, \ldots, r_m : \phi$$

is *alpha of degree* n, the list (finite sequence) $\langle t_1, \ldots, t_n \rangle$ is the *target list* and ϕ is the *qualificator* (qualified expression).

(vii) *Terms* are in addition the expressions

$$f(\alpha),$$

where α is alpha and f is some $aggregate function$.

3. Computational Model for R-Calculus

The draft of development and interpretation for C-calculus above has the inadequacies caused by:

complexity and insufficient clearness of syntax constructions which, besides our desire, contain some heuristical reasons;

complexity for range evaluations (domains) where the variable range.

These circumstances are the source for unpleasant loss of effectiveness when C-calculus is implemented. This gives rise to development of algorithms for query optimizing.

An outline for R-calculus will be given below which has the following features to deal with *events* or *knowledge stages*.

1) R-calculus, as a rule, should be used as an *extensional* language giving rise to data base queries. In this case the cross referencing points, or *assignments* $k \in Asg$ are used, i.e. the *data base configurations* are indicated explicitly. Whenever the cross referencing points are used implicitly then R-calculus obtains typical intensional features. This is analogous to known *frame algebra*. Moreover, the intentional R-calculus with quantifiers and aggregate functions, as will be shown,is the same as *frame algebra*with both usual and arithmetical quantifiers.

2) Both intensions and extensions, i.e. the *instantiations*, of expressions in R-calculus have an ordering as a structure. The intentions are ordered along to ISA-hierarchy and the extensions are ordered along the induced relation of a partial order.

3) The computational model for R-calculus has the ranges for variables both for basic and derived types, thus, giving rise to data base conceptual design.

4) Certainly, the C-calculus is distinct from R-calculus, and whenever a data base developer deals with the dynamical problem domains then he applies and prefers the more conceptual transparent R-calculus.

The features listed above will be given as follows in detailed and closed manner. They are ramified as a computational model for R-calculus and are in many details analogous to computational model for R-algebra. This is usually a start point for developing the reduction algorithms both in 'calculus – algebra' direction and 'algebra – calculus'.

3.0.1. Alphabet

$\star\ (\ ,\ )\ \lambda\ \wedge\ \vee\ \supset\ \equiv\ |\ \nabla\ K\ S\ \neg$

3.1. Type Symbols

The type symbols emulate attributes. Whenever σ, τ are nonempty words in alphabet $\star$ then (σ, τ) is typed symbol. The symbol $\star$ is a type symbol be the definition.

3.2. Variables

Whenever α is nonempty word in alphabet ∇, then $|\alpha|$ is a variable. The variables are denoted by x, y, z, v, possibly, with indices. Any type symbol is assigned to a countable set of variables. In further, term 'type' or 'attribute' is used instead of 'type symbol'. A variable with the assigned type σ is denoted by x_σ^n, or $x^n : \sigma$, or $x_n : \sigma$.

3.3. Objects

$\neg$, $\wedge$, $\vee$ and variables are declared as objects. Let a and b be the objects. Then (ab) and $(\lambda x a)$ are the objects as well. To save writing the parentheses could be omitted in case they can be reveal by the rule of 'association to the left

i.e. $a(bc)d \rightleftharpoons ((a(bc))d)$. The symbol $\rightleftharpoons$ is used as 'equals by the definition', symbol $=$ is used as 'equals graphically', and symbol $\not\equiv$ is used as 'differs graphically'.

$\lambda(x).a \rightleftharpoons \lambda x.a \rightleftharpoons \lambda xa$; $(ab) \rightleftharpoons a(b)$;

$\lambda x_1 \ldots x_n.a \rightleftharpoons \lambda(x_1, \ldots, x_n).a \rightleftharpoons \lambda x_1(\lambda x_2 \ldots x_n.a)$, $n > 1$.

The result in substituting variable x by an object b within the object a is denoted as $[b/x]a$.

3.4. A Standard Class of Variables

Let variable be treated as *indeterminant*. Class X of variables is determined as a *standard* class if for any set x_1, x_2, $\ldots x_{n-1}$, x_n of X there is variable y of X, distinct from $x_1, \ldots x_{n-1}, x_n$. Let $\mathfrak{N}$ be the class of objects, X be the standard class of variables, $X \subseteq \mathfrak{N}$, i.e. X is a subclass of the class $\mathfrak{N}$.

3.5. A Standard Triple

Let $\mathbb{C}[x]$ be the standard class of variables. They will be referred as the *individual* variables below. Let $\mathbb{P}[y]$ be nonempty nonintersecting with $\mathbb{C}[x]$ class of variables. They will be referred as the *predicate* variables. Let $\mathfrak{N}$ be the class of objects, $\mathbb{C}[x] \subseteq \mathfrak{N}$. Such a triple of classes will be referred as the *standard triple*:

$$(\mathbb{C}[x],\ \mathbb{P}[y],\ \mathfrak{N}).$$

3.6. Type Assignment

The set of free in an object d variables is denoted by $FV(d)$ and the set of bound in d variables is denoted by $BV(d)$. The *terms* as the following objects:

1) (*variable*) variable

$$x_n : \delta$$

is a term of type δ;

2) (*application*) if d is a term of type (δ, Δ), e is a term of type δ, then $(d\,e)$ is a term of type Δ:

$$\frac{d : (\delta, \Delta) \qquad e : \delta}{(d\,e) : \Delta}.$$

In this case the definition

$$\begin{aligned} FV((d\,e)) &\rightleftharpoons FV(d) \cup FV(e), \\ BV((d\,e)) &\rightleftharpoons BV(d) \cup BV(e) \end{aligned}$$

determines sets of both free $FV(\ldots)$ and bound $BV(\ldots)$ variables in the *application* $(d\,e)$. (Here: '$(d\,e)$' is read as 'd is applied to e', where d is observed as a function for the argument e, hence $d\,e$ is the result in applying a function d to the argument e.)

3) (*abstraction*) if d is a term of type Δ, y is a variable of type δ then (λyd) is a term of type (δ, Δ):

$$\frac{y : \delta \qquad d : \Delta}{(\lambda yd) : (\delta, \Delta)}$$

and

$$\begin{aligned} FV((\lambda yd)) &\rightleftharpoons FV(d) - \{y\}, \\ BV((\lambda yd)) &\rightleftharpoons BV(d) \cup \{y\}. \end{aligned}$$

(Here: '(λyd)' is a function from δ to Δ.)

A set of all the terms is denoted by Tm.

3.7. Formulae

Let $(\mathbb{C}[x],\ \mathbb{P}[y],\ \mathfrak{N})$ be the standard triple. A notion of *formula* is defined by induction as follows:

i) if y is a predicate variable and $x_1, \ldots, x_n$ are the individual variables for $n \geq 0$ then the object

$$y\,x_1 \ldots x_n = y(x_1, \ldots, x_n)$$

is an *elementary* formula;

ii) if a is an elementary formula, x is an individual variable, and $b \in \mathfrak{N}$, then

$$[b/x]a$$

is the elementary formula;

iii) any elementary formula is a formula;

iv) if a and b are the formulae then the objects $\supset a\,b$, $\wedge a\,b$, $\vee a, b$, $\equiv a\,b$ are the formulae;

v) if a is a formula, x is an individual variable, then the objects $\neg a$, $\forall \lambda x.a$, $\exists \lambda x.a$ are the formulae.

Let $\mathbb{C}[x]$ be a standard class of the variables. The result of substituting a variable $y \in \mathbb{C}[x]$ by the object b within the object a is written as

$$[b/y]a,$$

and in case $a = \lambda t.c$, $t \in FV(b) \cup BV(b)$, $t \not\equiv y$, $y \in FV(c) \cup BV(c)$ the following is assumed

$$[b/y]a \rightleftharpoons \lambda z.[b/y][z/t]c,$$

taking $z \in \mathbb{C}[x]$, $z \not\equiv y$, $z \notin FV(b) \cup BV(b) \cup FV(c) \cup BV(c)$.

If $(\mathbb{C}[x],\ \mathbb{P}[y],\ \mathfrak{N})$ is a standard triple of classes, then a substitution is meant as the $\mathbb{C}[x]$-substitution. If $P_n \in \mathbb{P}[y]$ then the predicate variable P_n is assumed as having been assigned the type $(\tau_1, \ldots, \tau_n)$. Here in the type expression the omitted parenteses are revealed by the association to the right, and $\tau_1, \ldots, \tau_n$ are the types of the argument places of n-ary predicate symbol P_n. Both the object b and variable y under the substitution $[b/y]a$ are assumed as having the same type.

3.8. Pre-Structure

If V_δ is a nonempty domain of *virtual objects* which is assigned to the type symbol δ and $\varepsilon_{\sigma\tau}$ is the mapping

$$\varepsilon_{\sigma\tau} : V_{(\sigma,\tau)} \times V_\sigma \to V_\tau$$

for any type symbols σ, τ, then the pair

$$(\{V_\sigma\}, \{\varepsilon_{\sigma\tau}\})$$

is called *pre-structure*. In addition, the extensionality (ext)

(ext): if $x, y \in V_{(\sigma,\tau)}$ and for any $z \in V_\sigma$
the equation $\varepsilon_{\sigma\tau}(x, z) = \varepsilon_{\sigma\tau}(y, z)$ is valid
then $x = y$

is requested.

An *assignment* (referencing point, possible world etc.) within pre-structure $(\{V_\sigma\}, \{\varepsilon_{\sigma\tau}\})$ is the function k, defined over all the variables and such that

$$x^n_\sigma(k) \in V_\sigma(k).$$

A set of all the assignments is denoted by Asg.

3.9. Structure

The structure is defined as a triple

$$(\{V_\sigma\}, \{\varepsilon_{\sigma\tau}, Val\}),$$

such that

$$(\{V_\sigma\}. \{\varepsilon_{\sigma\tau})$$

is a pre-structure, Val is the evaluation mapping

$$Val : Tm \to (\bigcup_{\sigma\in\mathcal{T}} V_\sigma)^{Asg},$$

where for set of all the types $\mathcal{T}$:

1) $Val(x_\sigma) \in V_\sigma^{Asg}$ and

$$Val(x_\sigma, k) = x_\sigma(k)$$

for any $k \in Asg$;

2) $Val((d\,e)) = (\varepsilon_{\sigma\tau}(Val(d), Val(e))) \subseteq V_\tau^{Asg}$ where subterms range respectively $Val(d) \subseteq V_{(\sigma,\tau)}^{Asg}$, $Val(e) \subseteq V_\sigma^{Asg}$ and

$$Val((d\,c), k) = \varepsilon_{\sigma\tau}(Val(d, k), Val(e, k))$$

for any $k \in Asg$, where d, e are the typed terms, $d : (\sigma, \tau)$, $e : \sigma$;

3) for any $c \in V_\sigma$

$$\varepsilon_{\sigma\tau}(Val((\lambda x.d), k), c(k)) = Val(d, k^x_c),$$

where $d : \tau$ is a term, $x : \sigma$, $y : \sigma$ are variables and

$$x(k^y_c) \rightleftharpoons \begin{cases} x(k) & \text{for } y \not\equiv x, \\ c(k) & \text{for } y \equiv x. \end{cases}$$

An explanation of this switch in the definition of evaluation Val is as follows: $Val((\lambda x.d), k)$ is such a function Φ that

$$\Phi : V_\sigma(k) \to V_\tau(k)$$

and for any $c(k) \in V_\sigma(k)$

$$\Phi(c(k)) = Val([c/x]d, k).$$

In above the value of $Val((\lambda x.d), k)$ is defined if and only if all the values $Val([c/x]d, k)$ are defined, and, in addition, the object (function) Φ has a type (σ, τ). Such a function Φ is mentioned as the *generator* for an object $\lambda x.d$.

3.10. Substitution

Let $\mathbb{C}[x]$ be a standard class of variables and define a function g from variables to terms such that $g(x)$ has the same type as x (*substitution*). A function $s(g)$ is defined so that every free occurrence of variable y in s is replaces by $g(y)$ (*multiple substitution*). More rigorously, let $[t/x]d$ be a substitution of the term t for every free occurrence of x in d, where d is a term, x is a variable, and t is a term of the same type as x. Then:

1) $[t/x]x = t$;
2) $[t/x]y = y$, where $x \not\equiv y$;
3) $[t/x](d\,e) = ([t/x]d)([t/x]e)$;
4) $[t/x](\lambda x.d) = \lambda x.d$;
5) $[t/x](\lambda y.d) = \lambda y.[t/x]d$, where $y \not\equiv x$.

The substitution g is called *regular* if for all the variables x and y

$$FV(g(x)) \cap BV(g(y)) = \varnothing.$$

3.11. Continuation of Val

Given a class of formulae and the boolean algebra $(B, \leq)$. For any n-ary predicate symbol $P \in \mathbb{P}[y]$ the n-ary function

$$\overline{P} : V_{\delta_1}^{Asg} \times \ldots V_{\delta_n}^{Asg} \to B^{Asg}, \quad B \rightleftharpoons \{true, false\},$$

is an *intension* of P and the n-ary function

$$\overline{P}(k) : V_{\delta_1}(k) \times \ldots V_{\delta_n}(k) \to B$$

is an *extension* of P, $k \in Asg$.

Accepting by the definition that

$$Val(P) \rightleftharpoons \overline{P}, \quad Val(P, k) \rightleftharpoons \overline{P}(k)$$

leads to the *concepts* of predicate symbols.

An analysis given below shows that the definition of a *concept* (of an *intensional*) is essentially the same as the definition of a *frame*.

The following gives the definition of a function that assigns *value* to the formulae. For $c \in V_\delta$ assume that

$$Val(c, k) \rightleftharpoons c(k)$$

for any assignment $k \in Asg$ (data base configuration), and

$$Pt_1 \ldots t_n \equiv P(t_1, \ldots, t_n),$$

and notations ϕ and ψ are used for arbitrary formulae.

1) $Val(P(t_1, \ldots, t_n)) = \varepsilon(\overline{P}, Val(t_1), \ldots, Val(t_n))$, where $\varepsilon(x_1, \ldots, x_{n+1})$ for corresponding x_1, ..., x_{n+1} is an abbreviation

$$\begin{aligned} \varepsilon(x_1, \ldots, x_{n+1}) &\equiv \varepsilon(\varepsilon(x_1, x_2), \ldots, x_{n+1}) \\ &\ldots \quad \ldots \\ &\equiv \varepsilon(\ldots(\varepsilon(x_1, x_2), \ldots), x_{n+1}) \end{aligned}$$

and every occurrence of ε means corresponding $\varepsilon_{\sigma\tau}$.

1') For $k \in Asg$ the equation

$$\begin{aligned} &Val(P(t_1, \ldots, t_n), k) = \\ &\quad = \varepsilon(\overline{P}(k), Val(t_1, k), \ldots, Val(t_n, k)) \end{aligned}$$

is such that 'an evaluation of the extensional for "entire" drops down to evaluations of the extensions for "parts" '.

2) $Val(\wedge\phi\psi) = \inf(Val(\phi), Val(\psi))$ for formulae ϕ, ψ.

2') $Val(\wedge\phi\psi, k) = \inf(Val(\phi, k), Val(\psi, k))$ for $k \in Asg$.

3) $Val(\vee\phi\psi) = \sup(Val(\phi), Val(\psi))$ for formulae ϕ, ψ.

3') $Val(\vee\phi\psi, k) = \sup(Val(\phi, k), Val(\psi, k))$ for $k \in Asg$.

4) $Val(\supset \phi\psi) = (Val(\phi) \Rightarrow Val(\psi))$ for formulae ϕ, ψ.

4') $Val(\supset \phi\psi, k) = (Val(\phi, k) \Rightarrow Val(\psi, k))$ for $k \in Asg$.

5) $Val(\exists\lambda x.\phi) = \sup\limits_{c \in V_\delta} \{Val([c/x]\phi)\}$.

5') $Val(\exists\lambda x.\phi, k) = \sup\limits_{c(k) \in V_\delta(k)} \{Val([c/x]\phi, k)\}$ for $k \in Asg$.

6) $Val(\forall\lambda x.\phi) = \inf\limits_{c \in V_\delta} \{Val([c/x]\phi)\}$.

6') $Val(\forall\lambda x.\phi, k) = \inf\limits_{c(k) \in V_\delta(k)} \{Val([c/x]\phi, k)\}$ for $k \in Asg$.

7) $Val(\lambda x_1 \ldots x_n.\phi)$ is a function $\overline{\phi}$ (*generator*, generating function):

$$\overline{\phi} : V_{\delta_1} \times \ldots \times V_{\delta_n} \to B^{Asg}$$

such that for any $c_i \in V_{\delta_i}$, $i = 1, \ldots, n$

$$\overline{\phi}(c_1, \ldots, c_n) = Val([c_1, \ldots, c_n/x_1, \ldots, x_n]\phi).$$

The value $Val(\lambda x_1 \ldots x_n.\phi)$ is determined if all the values

$$Val([c_1, \ldots, c_n/x_1, \ldots, x_n]\phi)$$

are determined, and, in addition the type for $\overline{\phi}$ is

$$\overline{\phi} : (\delta_1, \ldots, \delta_n, B^{Asg}),$$

where B is a boolean type.

7') Similar considerations give

$$\overline{\phi}(k) : V_{\delta_1}(k) \times \ldots \times V_{\delta_n}(k) \to B.$$

The rest of formulae are derived by the obvious transformations.

8) $Val(\neg\phi) \in B^{Asg}$.

8') $Val(\neg\phi, k) = true$ if and only if $Val(\phi, k) = false$ for fixed configuration $k \in Asg$.

3.12. Connection with Boolean Algebra

Consideration here will be done using schemata for R-algebra.

Let $\mathcal{T}$ be a set of all the *types* (attributes) for given some set of *basic types*. Every type symbol δ has been assigned with a set V_δ. Certainly, the *concepts* of types and *virtual* domains V_δ are used.

Let determine a set V as

$$V \rightleftharpoons \bigcup_{\delta \in \mathcal{T}} V_\delta$$

and use its *power set* $\mathcal{P}(V)$ with associate inclusion relation $\subseteq$:

> for f, $g \in \mathcal{P}(V)$ assume that $f \subseteq g$ if and only if membership $c \in f$, for any c, implies $c \in g$.

This consideration is intensional in its nature. Now consider the pair

$$(\mathcal{P}(V), \subseteq)$$

which is a *complete boolean algebra*.

The following theorem gives the needed details for instantiations in a computational model.

Theorem 3.1. *If ϕ, ψ, χ are formulae ($\mathbb{C}$ [x], $\mathbb{P}$[y], $\mathfrak{N}$)-formulae) then $\overline{\phi}$, $\overline{\psi}$, $\overline{\chi}$ are corresponding elements in boolean algebra, $\overline{\phi}$, $\overline{\psi}$, $\overline{\chi} \in \mathcal{P}(V)$ and $\overline{\phi} \rightleftharpoons Val(\lambda x_1 \ldots x_l.\phi)$, $\overline{\psi} \rightleftharpoons Val(\lambda y_1 \ldots y_m.\psi)$, $\overline{\chi} \rightleftharpoons Val(\lambda z_1 \ldots z_n.\chi)$, $l, m, n \geq 0$.*

Proof. In this case $\overline{\phi}$, $\overline{\psi}$, $\overline{\chi}$ in the configuration $k \in Asg$ generate a set of substitutions for which every of the formulae ϕ, ψ, χ have values *true*.

1) Let $(\mathbb{C}[x], \mathbb{P}[y], \mathfrak{N})$ be a standard triple, where $\mathbb{C}[x]$ is a class of individual variables, $\mathbb{P}[y]$ is a class of predicate variables, $\mathfrak{N}$ is a class of objects, where $\mathbb{C}[x] \subseteq \mathfrak{N}$ for $l = m = n = r \geq 0$ and types of variables x_i, y_i, z_i are the same for any $i = 1, \ldots, r$.

An inclusion

$$Val(\lambda x_1 \ldots x_r.\phi,\ k) \subseteq Val(\lambda y_1 \ldots y_r.\phi,\ k)$$

is denoted by

$$\overline{\phi}(k) \leq \overline{\psi}(k).$$

Note that in case of *concepts* the equation for ISA is immediately derived:

$$\overline{\phi}^{Asg}\ ISA\ \overline{\psi}^{Asg},$$

and if $x_1, \ldots, x_r \in FV(\phi)$ and $y_1, \ldots, y_r \in FV(\psi)$ then formulae ϕ and ψ are called as similar, and this is identical to the definition of *ISA-similar frames*.

2) A verification of the properties for boolean algebra is analogous (excepting the notations) to its verification in case of R-algebra. An assumption of one-to-one correspondence between $\overline{P}$ and $\overline{\phi}$, $\overline{Q}$ and $\overline{\psi}$, $\overline{R}$ and $\overline{\chi}$.

3) Let now $r \leq min\{l, m, n\}$ and the variables $x_{i\nu}$, $y_{j\nu}$, $z_{k\nu}$ have the same types τ_ν for $\nu = 1, 2, \ldots, r$. Let $v_1, \ldots, v_r$ are the variables distinct from variables x_α, y_β, z_γ for $\alpha = 1, \ldots, l$; $\beta = 1, \ldots, m$; $\gamma = 1, \ldots, n$, but the type of v_ν is the same as type of $x_{i\nu}$, $y_{j\nu}$, $z_{k\nu}$.

Consider the functions

$$\begin{aligned}
\overline{\phi}_r &\rightleftharpoons Val(\lambda v_1 \ldots v_r.[v_1, \ldots, v_r/x_{i1}, \ldots x_{ir}]\phi),\\
\overline{\psi}_r &\rightleftharpoons Val(\lambda v_1 \ldots v_r.[v_1, \ldots, v_r/y_{j1}, \ldots y_{jr}]\psi),\\
\overline{\chi}_r &\rightleftharpoons Val(\lambda v_1 \ldots v_r.[v_1, \ldots, v_r/z_{k1}, \ldots z_{kr}]\chi),
\end{aligned}$$

each of them has the type

$$(\tau_1, \ldots, \tau_r, B).$$

It could be shown that the conditions for boolean algebra are now valid, i.e. $\overline{\phi}_r\ \overline{\psi}_r\ \overline{\chi}_r$ are the elements of boolean algebra. Note that *conceptual* consideration leads immediately to a conclusion: ISA-similar frames generate the boolean algebra.

3.13. Connection with Algebra of Relations

Let ϕ and ψ be the formulae. Formula ϕ is mentioned as l-ary if $x_1, \ldots, x_l \in FV(\phi)$ and formula ψ is mentioned as n-ary if $z_1, \ldots, z_n \in FV(\psi)$, $l, n \geq 0$. Let ϕ is l-ary formula and ψ is n-ary formula, $(\mathbb{C}[x], \mathbb{P}[y], \mathfrak{N})$ is a standard triple, $\overline{\phi}_m$ and $\overline{\psi}_m$ are generating functions, $m \leq \min\{l, n\}$:

$$\begin{aligned}
&\overline{\phi}_m \rightleftharpoons\\
&\rightleftharpoons Val(\lambda v_1 \ldots v_m.[v_1, \ldots, v_m/x_{i1}, \ldots, x_{im}]\phi_m,\ k)\\
&= Val(\lambda v_1 \ldots v_m.\phi_m(v_1) \ldots (v_m),\ k)\\
&= \overline{\psi}_m(\overline{v}^1_{\tau 1}) \ldots (\overline{v}^m_{\tau m})(k),\\
&\overline{\psi}_m \rightleftharpoons\\
&\rightleftharpoons Val(\lambda v_1 \ldots v_m.[v_1, \ldots, v_m/z_{j1}, \ldots, z_{jm}]\psi_m,\ k)\\
&= Val(\lambda v_1 \ldots v_m.\psi_m(v_1) \ldots (v_m),\ k)\\
&= \overline{\psi}_m(\overline{v}^1_{\tau 1}) \ldots (\overline{v}^m_{\tau m})(k).
\end{aligned}$$

Assume that

$$\overline{\tau}(k) = |\tau, k|$$

is used for denoting a class of elements from $V_\tau(k)$. The notation

$$|\tau, k| \subseteq V_\tau(k)$$

is referred as a *domain for type τ in configuration $k \in Asg$*. The functions $\overline{\phi}_m(k)$ and $\overline{\psi}_m(k)$ are the mappings:

$$\begin{aligned}
\overline{\phi}_m(k) &: |\tau_1, k| \times \ldots \times |\tau_m, k| \to B,\\
\overline{\psi}_m(k) &: |\tau_1, k| \times \ldots \times |\tau_m, k| \to B,
\end{aligned}$$

such that for any $\overline{c}_i \in |\tau_i, k|$, $i = 1, \ldots, m$,

$$\begin{aligned}
&\overline{\phi}_m(\overline{c}_1) \ldots (\overline{c}_m)(k) \in B,\\
&\overline{\psi}_m(\overline{c}_1) \ldots (\overline{c}_m)(k) \in B.
\end{aligned}$$

The generating functions $\overline{\phi}_m(k)$ and $\overline{\psi}_m(k)$ are referred as *relations* (*target* relations, R-relations etc.) in configuration k defined on a cartesian product of the domains (ranges of virtual objects) $|\tau_1, k| \times \ldots \times |\tau_m, k|$:

$$\begin{aligned}
\overline{\phi}_m(\overline{c}_1) \ldots (\overline{c}_m) &= Val([c_1, \ldots, c_m/v_1, \ldots, v_m]\\
&\qquad [v_1, \ldots, v_m/x_{i1}, \ldots, x_{im}]\phi, k)\\
\overline{\psi}_m(\overline{c}_1) \ldots (\overline{c}_m) &= Val([c_1, \ldots, c_m/v_1, \ldots, v_m]\\
&\qquad [v_1, \ldots, v_m/z_{j1}, \ldots, z_{jm}]\psi, k).
\end{aligned}$$

The instantiation above will be denoted by

$$\begin{aligned}
&\overline{\phi}_m : (\tau_1, \ldots, \tau_m)(k),\\
&\overline{\psi}_m : (\tau_1, \ldots, \tau_m)(k),
\end{aligned}$$

where $\tau_1, \ldots, \tau_m$ are *attributes*. Note that the results above are similar to those for R-algebra besides the term 'predicate' which is replaced by the term 'formula'.

The definitions for operation of intersection INT, union $UNION$, and difference $DIFF$ on generating functions (in case of formulae) are similar to those for relations (predicates).

DIFF: Whenever for $k \in Asg$

$$\begin{aligned}
\inf(\overline{\phi}_m(k), \overline{\psi}_m(k)) &= \bot(k) = 0.\\
\sup(\overline{\psi}_m(k), \overline{\psi}_m(k)) &= \top(k) = 1.
\end{aligned}$$

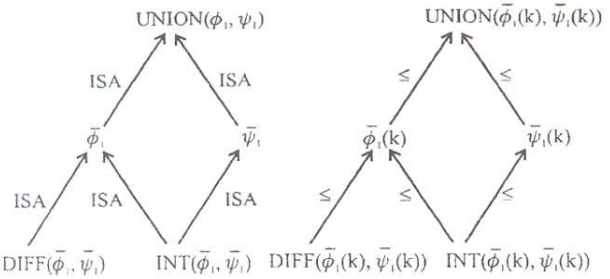

Figure 1: Boolean algebras: intensions (ISA-algebra) and extensions.

then intension (frame) $\overline{\psi}_m$ determines the *compliment* for intension (frame) $\overline{\phi}_m$,

$$COMPL(\overline{\phi}_m)^{Asg} \rightleftharpoons \overline{\psi}_m^{Asg}.$$

Note that

$$\begin{array}{rl} & COMPL(IMPL(\overline{\phi}_m, \overline{\psi}_m)) \\ = & \inf(\overline{\phi}_m, COMPL(\overline{\psi}_m)) \\ \rightleftharpoons & DIFF(\overline{\phi}_m, \overline{\psi}_m) \end{array}$$

(*difference* of frames).

INT: Determine that

$$INT(\overline{\phi}_m, \overline{\psi}_m) \rightleftharpoons \inf(\overline{\phi}_m, \overline{\psi}_m)$$

(*intersection*, conjunction of frames).

UNION: Define that

$$UNION(\overline{\phi}_m, \overline{\psi}_m) \rightleftharpoons \sup(\overline{\phi}_m, \overline{\psi}_m)$$

(*union*, disjunction of frames).

The proof that definitions of $DIFF$, INT, $UNION$ above hold in boolean algebra.

Example 3.1. *Let*

$$\begin{array}{rcl} \overline{\phi}_1 & \rightleftharpoons & \{c_1, c_2, c_3, c_5\}, \\ \overline{\psi}_1 & \rightleftharpoons & \{c_2, c_4, c_6, c_8\}. \end{array}$$

Then

$$\begin{array}{rcl} DIFF(\overline{\phi}_1(k), \overline{\psi}_1(k)) & = & \{c_1, c_3, c_5\} \\ INT(\overline{\phi}_1(k), \overline{\psi}_1(k)) & = & \{c_2\} \\ UNION(\overline{\phi}_1(k), \overline{\psi}_1(k)) & = & \{c_1, c_2, c_3, c_4, c_5, c_6, c_8\}. \end{array}$$

A sample of both intensional boolean algebra (ISA-algebra) and extensional boolean algebra are given in Fig. 1.

3.14 **Derived Types – Attributes.**

Let

$$\phi : (\delta_1, \ldots, \delta_l)$$

be l-ary formula and

$$\psi : (\Delta_1, \ldots \Delta_n)$$

be n-ary formula, where ϕ and ψ are ($\mathbb{C}[x]$, $\mathbb{P}[y]$, $\mathfrak{N}$)-formulae.

Define the generating function as follows:

$$\begin{array}{rcl} \overline{\phi}(\overline{x}_i) & \rightleftharpoons & Val(\lambda x_i.\phi)^{Asg},\ 0 \le i \le l, \\ \overline{\psi}(\overline{x}_j) & \rightleftharpoons & Val(\lambda x_j.\psi)^{Asg},\ 0 \le j \le n. \end{array}$$

In addition, for the definitions

$$\begin{array}{rcl} |\delta_i', k| & \rightleftharpoons & \inf(|\delta_i, k|,\ \overline{\phi}(\overline{x}_i)(k)), \\ |\Delta_j', k| & \rightleftharpoons & \inf(|\Delta_j, k|,\ \overline{\psi}(\overline{x}_j)(k)), \end{array}$$

following restrictions, written as the abbreviations, are valid:

$$\begin{array}{rclcrcl} |\delta_i', k| & \subseteq & |\delta_i, k|, & & |\delta_i', k| & \subseteq & \overline{\phi}(\overline{x}_i)(k), \\ |\Delta_j', k| & \subseteq & |\Delta_j, k|, & & |\Delta_j', k| & \subseteq & \overline{\psi}(\overline{x}_j)(k), \end{array}$$

and, without ambiguity for known $k \in Asg$:

$$\begin{array}{rclcrcl} \delta_i' & \le & \delta_i, & & \delta_i' & \le & \overline{\phi}(\overline{x}_i) \\ \Delta_j' & \le & \Delta_j, & & \Delta_j' & \le & \overline{\psi}(\overline{x}_j). \end{array}$$

Here: δ_i, Δ_j are *generic types*, "polynomials" $\overline{\phi}(\overline{x}_i)$, $\overline{\psi}(\overline{x}_j)$ are *restrictions* (relationships), and

$$\begin{array}{rcl} \delta_i' & \rightleftharpoons & \inf(\delta_i,\ \overline{\phi}(\overline{x}_i)), \\ \Delta_j' & \rightleftharpoons & \inf(\Delta_j,\ \overline{\psi}(\overline{x}_j)), \end{array}$$

are *derived types* under restrictions $\overline{\phi}(\overline{x}_i)$ and $\overline{\psi}(\overline{x}_i)$ respectively, where type of $\overline{x}_i$ is δ_i and type of $\overline{x}_j$ is Δ_j. The domains (sets)

$$\begin{array}{rcl} |\delta_i', k| & \subseteq & V_{\delta_i}(k), \\ |\Delta_j', k| & \subseteq & V_{\Delta_j}(k) \end{array}$$

are *derived domains*.

The derived domains defined as above are elements of boolean algebra. Of course, every derived domain corresponds to m-ary generating function which are the elements of algebra (algebra of generating functions, algebra of frames etc.). This is a "conceptual" algebra, and its m-ary operations are defined by a straightforward accommodation of the definitions above.

Note that the dual definitions of derived domains arise by replacing 'inf' by 'sup' and changing the directions of relations $\subseteq$ and $\le$ to the opposite one.

3.15. Connection with Relational Calculus (C-calculus by E.F. Codd).

Below the approach based on generating functions is applied to conform the computational model of C-calculus.

Assume that the following generating functions are defined:

$$\begin{aligned} \overline{\phi}_l : (\delta_1, \ldots, \delta_l) &\rightleftharpoons Val(\lambda x_1 \ldots x_l.\phi)^{Asg}, \\ \overline{\psi}_m : (\Delta_1, \ldots, \Delta_m) &\rightleftharpoons Val(\lambda z_1 \ldots z_m.\psi)^{Asg}. \end{aligned}$$

Of course, for $k \in Asg$

$$\begin{aligned} \overline{\phi}_l(k) &\subseteq |\delta_1,\ k| \times \ldots \times |\delta_l,\ k|, \\ \overline{\psi}_m(k) &\subseteq |\Delta_1,\ k| \times \ldots \times |\Delta_m,\ k|; \end{aligned}$$

the functions $\overline{\phi}_l$ and $\overline{\psi}_m$ are of types $(\delta_1, \ldots, \delta_l)$ and $(\Delta_1, \ldots, \Delta_m)$ respectively with the corresponding associated domains $|(\delta_1, \ldots, \delta_l),\ k|$ and $|(\Delta_1, \ldots, \Delta_m),\ k|$.

More rigorously, if the derived types are used, the following definitions are valid:

$$\begin{aligned} (\delta_1, \ldots, \delta_l)' &\rightleftharpoons \inf((\delta_1, \ldots, \delta_l),\ \overline{\phi}_l), \\ (\Delta_1, \ldots, \Delta_l)' &\rightleftharpoons \inf((\Delta_1, \ldots, \Delta_m),\ \overline{\psi}_m). \end{aligned}$$

Now the expressions of C-calculus which is sensitive to knowledge stages will be represented.

1) Given a schemata $\langle R_1, \ldots, R_N \rangle$, use the standard triple $(\mathbb{C}[x], \mathbb{P}[x], \mathfrak{N})$.

2) If P_j for $j = 1, \ldots, N$ are predicate and $x_1, \ldots, x_n$ are individual variables for $n \geq 0$, then object

$$P_j(x_1, \ldots, x_n)$$

is an *elementary formula* referred as *range term*.

3) Individual variables $x_i : \delta_i$ and individual constants $c_i : \delta_i$ are included into C-calculus, each using its own type.

4) If $\theta \in \{=, \neq, <, \leq, \geq, >\}$, when θ is used separately for each pair of types[1], then

$$\alpha\theta\beta \equiv \theta\alpha\beta$$

is an elementary formula referred as *join term* for any α, β which are either individual variables or individual constants. Any of combinations 'variable – variable', 'constant – variable', 'variable – constant', 'constant – constant' are allowed.

5) Define well formed formulae (wff) of C-calculus:

(i) Every elementary formula is wff.

(ii) If Γ is wff, then $\neg\Gamma$ is wff.

(iii) If Γ_1, Γ_2 are wff, then $\Gamma_1 \wedge \Gamma_2$ and $\Gamma_1 \vee \Gamma_2$ are wff.

(iv) If Γ is wff and a set of variables $\{x_1, \ldots, x_n\} \in FV(\Gamma)$, then $\exists^n \lambda x_1 \ldots x_n.\Gamma$ is wff and $\forall^n \lambda x_1 \ldots x_n.\Gamma$ is wff, where

$$\begin{aligned} \exists^n \lambda x_1 \ldots x_n.\Gamma &\rightleftharpoons \exists \lambda x_1.(\exists^{n-1} \lambda x_2 \ldots x_n.\Gamma), \\ \forall^n \lambda x_1 \ldots x_n.\Gamma &\rightleftharpoons \forall \lambda x_1.(\forall^{n-1} \lambda x_2 \ldots x_n.\Gamma), \end{aligned}$$

for $n \geq 1$, $BV(Q^n \lambda x_1 \ldots x_n.\Gamma) = BV(\Gamma) \cup \{ x_1, \ldots, x_n \}$, $Q^n \in \{\exists^n, \forall^n\}$.

6) Assume that Δ is some wff for which $\{x_1, \ldots, x_n\} \subseteq FV(\Delta)$. Commonly used notion of *subformula*, i.e. those fact that Δ is subformula of wff Γ, is denoted by $\Gamma(\Delta)$. The means of quantification are denoted by $Q^n \lambda x_1 \ldots x_n.\Gamma(\Delta)$.

7) The *range* quantifiers are in use by the definitions

$$\begin{aligned} \exists^n \Gamma(\Delta) &\rightleftharpoons \exists^n \lambda x_1 \ldots x_n.(\Gamma \wedge \Delta), \\ \forall^n \Gamma(\Delta) &\rightleftharpoons \forall^n \lambda x_1 \ldots x_n.(\neg\Gamma \vee \Delta). \end{aligned}$$

Here the logical connectives and predicates θ are written in infix form whenever there is no ambiguity.

8) *Range separable* wff are those like

$$U_1 \wedge U_2 \wedge \ldots \wedge U_n \vee V,$$

where

a) $n \geq 1$;

b) all of the U_i, $i = 1, \ldots, n$ are *proper range* wffs. Note that U_i is a proper range wff if its set of variables $\{x_{i1}, \ldots, x_{il}\}$ for $l \geq 1$ is the same as its set of free variables $FV(U_i)$;

c) formula V is either empty or:
all its quantifiers are ranged quantifiers;
$FV(V) \subseteq FV(U_1) \cup \ldots \cup FV(U_n)$;
V does not contain the range terms.

9) Alpha-expressions, or expressions of C-calculus, are objects

$$\begin{aligned} &\lambda x_1 \ldots x_m.W \equiv \\ &\quad \equiv \lambda(x_1, \ldots, x_m).W \rightleftharpoons (x_1, \ldots, x_m) : W, \end{aligned}$$

where

W is range separable wff;

$\{x_1, \ldots, x_m\} \subseteq FV(W)$.

10) Add to a class of join terms the terms

$$f_1(\alpha_1)\theta f_2(\alpha_2),$$

where

α_1, α_2 are alpha-expressions;

f_1, f_2 are function symbols (aggregate functions), $f_i \in \{SUM, MAX, MIN, AVG, COUNT, \ldots\}$.

[1] Assume that there is a family of binary predicates θ.

Consider entities $f_i(\alpha_i)$ as added to a class of alpha-expressions.

3.16. Evaluation of Alpha-Expressions

Now, within framework a computational model, alpha-expressions are ordinary objects. In further considerations the aggregate functions are not involved. However, taking them into account does not add the principle difficulties. Conformation of the further results, given below and based on 1)÷9) above, to the extended alpha-expressions with aggregate functions is straightforward and uses some syntax complications as outlined in 10).

3.16.1. Relationship with Algebra of Relations

The relationship with algebra of relations, i.e. consideration of R-algebra for alpha-expressions from C-calculus is easily obtained by the *generating functions*.

First of all, for alpha-expressions $\lambda(x_1, \ldots, x_l).W_1$ and $\lambda(z_1, \ldots, z_n).W_2$ the corresponding generating functions are as follows:

$$\begin{aligned}
&\overline{\phi}_m(k) \rightleftharpoons \\
&\rightleftharpoons Val(\lambda v_1 \ldots v_m.[v_1, \ldots, v_m/x_{i1}, \ldots x_{im}]W_1,\ k) \\
&= Val(\lambda v_1 \ldots v_m.W_1(v_1) \ldots (v_m),\ k) \\
&= \overline{W}_{1,m}(\overline{v}^1_{\tau 1}) \ldots (\overline{v}^m_{\tau m})(k), \\
&\overline{\psi}_m(k) \rightleftharpoons \\
&\rightleftharpoons Val(\lambda v_1 \ldots v_m.[v_1, \ldots, v_m/z_{j1}, \ldots z_{jm}]W_2,\ k) \\
&= Val(\lambda v_1 \ldots v_m.W_2(v_1) \ldots (v_m),\ k) \\
&= \overline{W}_{2,m}(\overline{v}^1_{\tau 1}) \ldots (\overline{v}^m_{\tau m})(k),
\end{aligned}$$

where $0 \leq m \leq \min(l, n)$, and $k \in Asg$.

Then apply the same reasons as in case of evaluation of expressions in R-calculus.

3.16.2. Relationship with Boolean Algebra and Derivation of Algebra of Frames

Relationship with boolean algebra and derivation of algebra of frames is established the same way as in case of R-calculus, including INT, $UNION$, $DIFF$.

3.16.3. Derived Attributes

All the apparatus of *derived attributes* over ISA-hierarchy can be obtained as well, resulting in R-algebra of expressions from C-calculus.

4. Conclusion

The feature of the computational model, described in this paper, is in explicit indication of the subject within the formal system via "referencing points" $k \in Asg$. Compare with intensional meaning of Asg as set of assignments, actual configurations of database, subjects, time, 'stages of knowledge', 'possible worlds' etc. This determines the computational model given here from the commonly used models for relational languages.

Acknowledgement

This research is supported in part by the Institute "Jurinfor-MSU" and by the Russian Foundation for Basic Research (RFBR) during the fulfillment of projects 04-07-90156, 05-01-00736 and implementation of the project 05-07-08012.

References

[1] Codd E.F. *The relational model for database management: version 2.* Addison-Wesley Publishing Company, 1990. – 538 p.

[2] Klug A. *Equivalence of relational algebra and relational calculus query languages having aggregate functions.* JACM, 1982, Vol. 29, No 3.

[3] Maier D., Ullman J.D., and Vardi M.Y. *On the foundations of the universal relational model.* ACM TODS, 1984, Vol. 9, No 2.

[4] Roussoupoulos N. *Semantic network model of data bases.* Ph.D. Thesis, Department of Computer Science, University of Toronto, 1977.

[5] Scott D.S. *Advice on modal logic.* Philosophical problems in logic. Some recent developments.- Lambert K. (ed.), Dordrecht; Holland: Reidel, 1970.

[6] Scott D.S. *Outline of a mathematical theory of computation.* Proceedings of the 4-th Annual Princeton conference on information sciences and systems, 1970.

[7] Scott D.S. *Lambda calculus: some models, some philosophy.* The Kleene Symposium. Barwise, J., et al.(eds.), Studies in Logic 101, North– Holland, 1980, pp.381-421.

[8] Scott D.S. *Relating theories of the lambda calculus.* Hindley J., Seldin J. (eds.) To H.B.Curry: Essays on combinatory logic, lambda calculus and formalism.- N.Y.& L.: Academic Press, 1980, pp. 403-450.

[9] Scott D.S. *Domains for denotational semantics.* LNCS, 140, 1982, pp. 577- 613.