Completeness Theorems for Program-oriented Algebra-based Logics of Partial Quasiary Predicates

Mykola S. Nikitchenko and Stepan S. Skylniak

Department of Theory and Technology of Programming
Taras Shevchenko National University of Kyiv
64, Volodymyrska Street, 01601 Kyiv, Ukraine

{nikitchenko, sssh}@unicyb.kiev.ua

Abstract. We provide motivation for developing and studying program-oriented logics of partial predicates. Such logics are algebra-based logics constructed in a semantic-syntactic style on the methodological basis that is common with programming; they can be considered as generalizations of traditional logics on classes of partial predicates that do not have fixed arity. Such predicates, called quasiary predicates, are defined over partial variable assignments (over partial data). We describe the hierarchy of different logics of quasiary predicates. For the constructed logics some laws of classical logic fail because of partiality of predicates and data. We construct sequent calculi for a number of defined logics and prove their soundness and completeness. The methods proposed can be useful for construction and investigation of logics for program reasoning.

Keywords: Partial predicate, partial logic, first-order logic, validity, sequent calculus, soundness, completeness.

1 Introduction

Mathematical logic is widely used for investigation of programs [1, 2]. Still, there are certain *discrepancies* between problems to be solved in this area and a logic in use. For example, such program properties as *partiality of functions, elaborated system of data types, behavioural non-determinism* etc. are difficult to investigate by traditional logic. To cope with such discrepancies we propose to construct *logics based directly on program models*.

To realize this idea we first construct models of programs using *composition-nominative approach* [3]. Principles of the approach (*development* of program notions *from abstract to concrete*, *priority of semantics*, *compositionality* of programs, and *nominativity* of program data) form a methodological base of program model construction. These principles specify program models as *composition-nominative systems*. Such a system may be considered as a triple of simpler systems: *composition*, *description*, and *denotation* systems. A *composition* system defines semantic aspects of programs, a *description* system defines program descriptions (syntactic aspects), and a *denotation* system specifies meanings (referents) of descriptions. We consider semantics of programs as partial functions over a class of data processed by programs; compositions are *n*-ary operations over functions. Thus, a composition system can be specified by two algebras: *data algebra* and *function algebra*. Function algebra is the main *semantic notion* in program formalization. *Terms* of this algebra define *syntax* of programs (description system), and ordinary procedure of term interpretation gives a *denotation* system.

The constructed formal program models form a base for developing of a rigorous mathematical formalism for reasoning about programs, in other words – *a program logic*. It is not possible to invent one universal program logic that would have all necessary properties. Therefore a hierarchy of logics, oriented on the hierarchy of program models, has to be developed. Here we briefly introduce such a hierarchy. Obtained logics are called *composition-nominative logics* (CNL).

It is important to admit that CNL better reflect program properties, but the opposite side of this feature is that the methods of logic investigation turn out to be more complicated. In this paper we continue our work on studying CNL focusing on the completeness problems for the first-order CNL.

The rest of the paper is structured as follows. In section 2 we give a motivational example of program formalization and logic construction. In section 3 we briefly introduce a hierarchy of logics of quasiary predicates. We define first-order pure CNL in section 4. In section 5 we consider properties of the consequence relation. In section 6 a sequent calculus for first-order pure CNL is defined; its soundness and completeness are demonstrated. In section 7 conclusions are formulated.

Proofs are omitted here and will be provided in an extended version of the paper.

2 Constructing Program Algebras and Logics: a Motivating Example

Let us consider the example language EL which is used here to demonstrate how program logics can be constructed. EL is similar to such languages as WHILE [4], IMP [5], etc.

The grammar of the language is defined as follows:

s::= x:=a | s_1 ; s_2| if b then s_1 else s_2 |while b do s | begin s end

a::= k | x | $a_1 + a_2$ | $a_1 - a_2$ | $a_1 * a_2$ | a_1 *div* a_2 | a_1 *mod* a_2 | (a)

b::=a_1= a_2 |a_1>a_2 | $b_1 \vee b_2$ |$\neg b$ | (b),

where:

- k ranges over integers Int={…, –2, –1, 0, 1, 2, …},
- x ranges over variables (names) V={ N, R, X, Y, Z, …}
- a ranges over arithmetic expressions *Aexpr*,
- b ranges over Boolean expressions *Bexpr*,
- s ranges over statements (programs) *Stm*.

As an example consider an EL-program *ES* for calculating a function $r = x^n$ ($n \geq 0$) using *Exponentiation by Squaring Algorithm*. In this program variables N, X, and R denote integer values n, x, and r respectively:

```
R:=1;
while N>0 do
  if (N mod 2)=1
      then begin R:=R*X; N:=N–1 end
      else begin X:=X*X; N:=N div 2 end
```

Starting from this example we construct program algebras of three forms:

- first, we define semantics of *ES* in the style of *denotational semantics*; as a result we obtain a *program algebra with n-ary mappings* oriented on EL;
- then we represent n-ary mappings by *quasiary mappings* obtaining a simpler *program algebra*;
- at last, we define a *general class* of quasiary program algebras. This class of algebras is a semantic base for *quasiary program logics*. It captures main program properties that are invariant of such programs' specifics as variable typing, interpreted predicates and functions, etc.

Analyzing the structure of the program we see that it is constructed from 1) symbols of *n-ary operations* (+, –, *, *div*, *mod*, =, >), 2) *Boolean* (N>0, (N *mod* 2)=1) and *arithmetic* (1, 2, N–1, N *div* 2, N *mod* 2, $R*X$, $X*X$) *expressions*, and 3) *statements* obtained with the help of structuring constructs such as assignment, sequence, selection, and loop. Note that *div* is partial on *Int*. To emphasize mapping's *partiality/totality* we write the sign $\xrightarrow{p}$ for partial mappings and the sign $\xrightarrow{t}$ for total mappings.

We use *denotational semantics* (see, e.g. [4, 5]) to formalize the meaning of program components. Semantic mapping is represented as $[\![.]\!]$. Three program components identified above determine three types of mappings called respectively *n-ary*, *quasiary*, and *bi-quasiary* mappings.

2.1 Classes of *n*-ary mappings

Symbols of arithmetic operations, relations, and Boolean connectives represent n-ary mappings defined on *Int* or on the set *Bool*={T, F} of Boolean values.

For our language EL we define the following types of n-ary mappings:

$$Fn^{n,Int}=Int^n \xrightarrow{p} Int,\ Pr^{n,Int}=Int^n \xrightarrow{p} Bool,\ Pr^{n,Bool}=Bool^n \xrightarrow{p} Bool,\ n\geq 0.$$

Using the same notation for language symbols of various types and mappings they represent, we can write that

$$+, -, *, div, mod: Fn^{2,Int};\ =,\ >: Pr^{2,Int};\ \vee: Pr^{2,Bool},\ \neg: Pr^{1,Bool}.$$

2.2 Classes of quasiary mappings

Quasiary mappings are defined over classes of states considered as sets of named values. For example, the expression R_*X specifies a function which given a state d of the form $[R \mapsto r, X \mapsto x]$, where r and x are integers, evaluates a value r_*x. Examples of states are $[X \mapsto 8, N \mapsto 4]$, $[X \mapsto 8, N \mapsto 4, R \mapsto 8]$, $[X \mapsto 8]$. In a state d a variable v can have a value (this is denoted $d(v)\downarrow$) or be undefined (denoted $d(v)\uparrow$); thus, $[X \mapsto 8, N \mapsto 4](X)\downarrow$ and $[X \mapsto 8, N \mapsto 4](R)\uparrow$. Formally, the set of states *State* is defined as the set ${}^{V}A=V \xrightarrow{p} Int$ of all partial mappings from V to *Int*.

Having described states we are able to represent formal semantics of Boolean and arithmetic expressions. Boolean expressions denote predicates (called *partial quasiary predicates*) of the set $Pr^{V, Int}=State \xrightarrow{p} Bool$; thus for $b \in Bexpr$ we have $[\![b]\!] \in Pr^{V, Int}$. Integer expressions denote functions (called *partial quasiary functions*) of the set $Fn^{V, Int}=State \xrightarrow{p} Int$; thus, for $e \in Aexpr$ we have $[\![e]\!] \in Fn^{V, Int}$. In the sequel we omit the term 'partial' for quasiary mappings. Since states are constructed with the help of naming (nominative) relation, they are also called *nominative sets*. Functions from $Fn^{V, Int}$ are called *ordinary functions* since their ranges are sets of atomic (non-structured) values.

To represent semantics of variables in arithmetic expressions we will use a *parametric denomination* (denaming) functions $'x$: $Fn^{V, Int}$. For a given program state, the function $'x$ returns the value of the variable x in that state. For instance, denomination function that yields the value of name N is denoted by $'N$. Such values may or may not be defined, so denomination functions are partial.

Semantics of integer numbers is treated as *quasiary constant functions*. Such functions are represented by constants $\boldsymbol{k}$ written in bold font; thus, $[\![1]\!] = \mathbf{1}$. It is clear that $\mathbf{1} \in Fn^{V, Int}$.

For specifying semantics of complex expressions special compositions called *superpositions* are used. Superposition $S_F^n : Fn^{n,Int} \times (Fn^{V,Int})^n \xrightarrow{t} Fn^{V,Int}$ of functions $g_1,\ldots, g_n$ into an n-ary function f^n is an operator such that $S_F^n(f^n, g_1,\ldots, g_n)(d) = f^n(g_1(d),\ldots,g_n(d))$ where d is a state. The same formula can be used for defining superposition $S_P^n : Pr^{n,Int} \times (Fn^{V,Int})^n \xrightarrow{t} Pr^{V,Int}$ of functions $g_1,\ldots, g_n$ into n-ary predicate p^n, and for superposition $S_B^n : Pr^{n,Bool} \times (Pr^{V,Int})^n \xrightarrow{t} Pr^{V,Int}$ of predicates $p_1,\ldots, p_n$ into an n-ary Boolean function f_B^n. Thus, $[\![N–1]\!] = S_F^2(–, 'N, \mathbf{1})$, $[\![N>0]\!] = S_P^2(>, 'N, \mathbf{0})$, $[\![(N \bmod 2)=1]\!] = S_P^2(=, S_F^2(mod, 'N, \mathbf{2}), \mathbf{1})$.

2.3 Classes of bi-quasiary mappings

Programs (statements) from *Stm* denote *bi-quasiary functions* (*program functions*) of the class $FPrg^{V,Int}=State \xrightarrow{p} State = {}^{V}Int \xrightarrow{p} {}^{V}Int$. Such functions are also called *bi-nominative*.

Semantics of structured statements is defined by the following compositions with conventional meaning:

1. *assignment composition* AS^x: $Fn^{V,Int} \xrightarrow{t} FPrg^{V,Int}$ (x is a parameter from V);
2. *composition of sequential execution* $\bullet$: $FPrg^{V,Int} \times FPrg^{V,Int} \xrightarrow{t} FPrg^{V,Int}$;
3. *conditional composition IF*: $Pr^{V,Int} \times FPrg^{V,Int} \times FPrg^{V,Int} \xrightarrow{t} FPrg^{V,Int}$;
4. *loop composition WH*: $Pr^{V,Int} \times FPrg^{V,Int} \xrightarrow{t} FPrg^{V,Int}$.

For instance, $[\![R:=R_*X; N:=N\text{-}1]\!] = AS^R(S_F^2(*, 'R, 'X)) \bullet AS^N(S_F^2(–, 'N, \mathbf{1}))$.

Note, that we define $\bullet$ by commuting arguments of conventional functional composition: $f \bullet g = g \circ f$.

2.4 Program algebra with *n*-ary mappings

The definitions introduced permit to conclude that the following *program algebra with n-ary mappings* oriented on ET has been constructed (we omit types of compositions):

$APn(V, Int) = < Fn^{2,Int}, Pr^{2,Bool}, Pr^{1,Bool}, Pr^{2,Int}, Pr^{V,Int}, Fn^{V,Int}, FPrg^{V,Int};$

$+, –, *, div, mod, =, >, \vee, \neg, \boldsymbol{k}, S_F^2, S_P^2, S_B^2, S_B^1, 'x, AS^x, \bullet, IF, WH >$.

Note that notation for parametric compositions (like denominations, assignments etc.) represents here classes of compositions. Thus, $'x$ represents the class of compositions for various x; $\boldsymbol{k}$ represents the class of integer constants treated as quasiary constant functions.

We would like to emphasize the fact that semantics of EL-programs (or EL-expressions) can be represented as terms of this algebra. This simplifies investigations of EL-programs because the constructed algebra completely specifies their semantics.

The term for EL-program *ES* is as follows:

$$AS^R(\mathbf{1}) \bullet WH(S_P^2(>,'N,\mathbf{0}), IF(S_P^2(=, S_F^2(mod,'N,\mathbf{2}),\mathbf{1}),$$

$$AS^R(S_F^2(*,'R,'X)) \bullet AS^N(S_F^2(-,'N,\mathbf{1})), AS^X(S_F^2(*,'X,'X)) \bullet AS^N(S_F^2(div,'N,\mathbf{2})))).$$

Note that this term and its sub-terms can denote partial mappings, as the function *div* and denomination functions can be undefined; also *WH* composition can be a source of undefinedness.

Having specified this algebra we can study properties of programs; this can be used in program reasoning. For example, it is possible to prove commutativity of the assignment statements $R{:=}R*X$ and $N{:=}N{-}1$ by proving in the algebra $APn(V, Int)$ the corresponding property of composition of sequential execution:

$$AS^R(S_F^2(*,'R,'X)) \bullet AS^N(S_F^2(-,'N,\mathbf{1})) = AS^N(S_F^2(-,'N,\mathbf{1})) \bullet AS^R(S_F^2(*,'R,'X))$$

We can also prove more general properties; say we can prove associativity of sequential execution of statements:

$$\text{begin } s_1 ; s_2 \text{ end}; s_3 \;=\; s_1; \text{begin } s_2 ; s_3 \text{ end}$$

by proving in the algebra $APn(V, Int)$ the corresponding property of composition of sequential execution: $f\bullet(g\bullet h)=(f\bullet g)\bullet h$ $(f, g, h \in Fn^{V,Int})$. Another example is the following distributivity property:

$$\text{begin if } b \text{ then } s_1 \text{ else } s_2 \text{ end}; s_3 = \text{ if } b \text{ then begin } s_1; s_3 \text{ end else begin } s_2; s_3 \text{ end}.$$

Its validity is based on the property of $APn(V, Int)$ that $IF(p,f,g)\bullet h=IF(p, f\bullet h, g\bullet h)$. (Here $f, g, h \in Fn^{V,Int}$, $p \in Pr^{V,Int}$.)

Still, the constructed program algebra with *n*-ary mappings looks overcomplicated; therefore we construct a *simpler algebra without n-ary mappings*. It is possible because *n*-ary mappings can be mimicked by quasiary mappings. Still, we should first split the set of operation symbols of this algebra into two parts: logical symbols, which have relatively type-independent interpretations, and non-logical (or descriptive) symbols, which represent specifics of the carriers. Logical symbols will be treated as compositions and descriptive symbols as quasiary mappings. Trivial inspection of definitions shows that symbols +, –, *, *div*, *mod*, =, > are descriptive symbols in the signature of $APn(V, Int)$. They are defined over integer numbers, and may be considered constants in this algebra. Other symbols may be considered logical.

2.5 Program algebra without *n*-ary mappings

The considerations described above open the second phase of program algebra development. To make the algebra simpler we can exclude from it the classes of *n*-ary functions and predicates, thus concentrating on logical symbols that are interpreted as compositions over nominative carriers $Pr^{V,Int}$, $Fn^{V,Int}$, and $FPrg^{V,Int}$. Still, *n*-ary functions and predicates can be represented in these classes of nominative mappings.

We explain the idea of representation of *n*-ary mappings on the example of binary multiplication function. First, we represent a pair (x_1, x_2) as a state $[1 \mapsto x_1, 2 \mapsto x_2]$, where 1 and 2 should be treated as standard variables that represent the arguments of a binary function symbol. This permits to treat multiplication as a quasiary function. Then, in order to avoid usage of standard names 1 and 2 and to obtain homogeneity of names we can introduce a parametric quasiary function $\boldsymbol{x*y}$ (printed in bold font) such that $\boldsymbol{x*y} = S_F^2(*, 'x, 'y)$; here x and y are parameters from V.

Therefore instead of binary functions we introduce parametric quasiary functions $\boldsymbol{x+y}$, $\boldsymbol{x-y}$, $\boldsymbol{x*y}$, $\boldsymbol{x\ div\ y}$, and $\boldsymbol{x\ mod\ y}$ over *Int*; also instead of relations we introduce new parametric quasiary predicates $\boldsymbol{x=y}$ and $\boldsymbol{x>y}$ (with x and y as parameters).

This step permits to represent every *n*-ary function defined over *Int* as a parametric quasiary function. But now, to represent the semantics of complex expressions we should introduce special superpositions $S_F^{v_1,\dots,v_n}$ (or $S_F^{\bar{v}}$) and $S_P^{v_1,\dots,v_n}$ (or $S_P^{\bar{v}}$), which are called *superpositions into quasiary function* and *predicate* respectively: $S_F^{v_1,\dots,v_n}(f^q, g_1,\dots, g_n)(d) = f^q(d\nabla[v_1 \mapsto g_1(d),\dots,v_n \mapsto g_n(d)])$ and $S_P^{v_1,\dots,v_n}(p^q, g_1,\dots, g_n)(d) = p^q(d\nabla[v_1 \mapsto g_1(d),\dots,v_n \mapsto g_n(d)])$, where $f^q \in Fn^{V,Int}$, $p^q \in Pr^{V,Int}$.

Intuitive meaning of these formulas is that we change in d the values of names $v_1,\ldots, v_n$ to $g_1(d),...,g_n(d)$ respectively (partiality should be taken into account). Thus, semantics, say, of the expression $R+(R_*X)$, can be represented as S_F^Y (***R+Y*** , ***R*∗*X***).

Now let us consider logical symbols $\vee$: $Bool^2 \xrightarrow{t} Bool$ and $\neg$: $Bool \xrightarrow{t} Bool$. We cannot directly represent them as quasiary predicates, therefore we advocate another approach. We will treat them as binary compositions over quasiary predicates (denoted by the same signs) $\vee$: $Pr^{V,Int}\times Pr^{V,Int} \xrightarrow{t} Pr^{V,Int}$ and $\neg$: $Pr^{V,Int} \xrightarrow{t} Pr^{V,Int}$. Such representations also provide better possibilities to work with partial predicates. For example, consider a Boolean expression $(R_*X)>R \vee (R>X)$. Its semantics in $APn(V, Int)$ is represented by the term $S_B^2(\vee, S_P^2(>, S_F^2(-, 'R, 'X), 'R), S_P^2(>, 'R, 'X)))$. But superposition into an n-ary mapping is strict: when one argument is not defined then the result is also undefined. This property restricts possibilities of construction of new partial predicates. For example, for Kleene's strong disjunction [6] it is allowed that one argument may be undefined if the other one is evaluated to true. When we represent connectives as compositions, we avoid the above considered difficulties.

Thus, we can now consider a simpler algebra – *the program algebra of quasiary predicates with constants* (*without n-ary mappings*):

$APQC(V, Int) = < Pr^{V,Int}, Fn^{V,Int}, FPrg^{V,Int};$

$$\boldsymbol{x-y}, \boldsymbol{x+y}, \boldsymbol{x*y}, \boldsymbol{n\ div\ m}, \boldsymbol{n\ mod\ m}, \boldsymbol{k}; \boldsymbol{x=y}, \boldsymbol{x>y}; \vee, \neg, S_F^{\bar{v}}, S_P^{\bar{v}}, 'x, AS^x, \bullet, IF, WH>.$$

In this algebra we have 8 *parametric descriptive symbols* considered as algebra's constants, and 9 *logical symbols*. Note, that all *logical symbols are treated as compositions*, possibly null-ary compositions (as in the case of denomination functions).

Semantics of *ES* program is represented by the following term of this algebra:

$$AS^R(\mathbf{1}) \bullet WH(\boldsymbol{N}>\mathbf{0}, IF((\boldsymbol{N\ mod}\ \mathbf{2})=\mathbf{1}, AS^R(\boldsymbol{R*X}) \bullet AS^N(\boldsymbol{N}-\mathbf{1}), AS^X(\boldsymbol{X*X}) \bullet AS^N(\boldsymbol{N\ div}\ \mathbf{2}))).$$

2.6 Program algebra without descriptive constants

The next step of constructing more "logical" algebras consists of eliminating from $APQC(V, Int)$ descriptive symbols ***x+y***, ***x–y***, ***x∗y***, ***x div y***, ***x mod y***, ***k***, ***x=y***, ***x>y*** with fixed interpretations. We obtain a *program algebra of quasiary predicates* $APQ(V, Int)$ *without constants*:

$$APQ(V, Int) = < Pr^{V,Int}, Fn^{V,Int}, FPrg^{V,Int}; \vee, \neg, S_F^{\bar{v}}, S_P^{\bar{v}}, 'x, AS^x, \bullet, IF, WH >.$$

As to eliminated descriptive symbols, we can instead consider sets *Ps*, *Fs*, and *FPs* of predicate, ordinary function, and program function symbols that do not have predefined interpretations, and consequently, can denote any quasiary predicate or function or bi-quasiary function.

The algebra $APQ(V, Int)$ is based on integer numbers, though it is clear that we can consider other data types, say *real*, *nat*, etc. Also, new operations over such data types can be considered. So, one can ask a question: what EL-program properties remain valid under type and operation variations?

2.7 General program algebras (with arbitrary classes of basic values)

Being interested in general laws of reasoning about programs we should make the next step and define compositions for any set A of basic (atomic) values. In this case we obtain the following *program algebra of quasiary mappings*:

$$APQ(V, A) = < Pr^{V,A}, Fn^{V,A}, FPrg^{V,A}; \vee, \neg, S_F^{\bar{v}}, S_P^{\bar{v}}, 'x, AS^x, \bullet, IF, WH >.$$

Symbols from *Ps*, *Fs*, and *FPs* are used to construct terms of this algebra. Properties of such terms are general properties because they should be valid under any interpretations of function and predicate symbols.

It means that we have constructed a class of quasiary program algebras (for various A), representing program semantics for languages with different domains. Such algebras may be called *general program models*; they form the semantic base for program logics.

For example, we can consider *equational program logics* by defining formulas of these logics as formal equalities of the form $t_1=t_2$, where t_1 and t_2 are terms of the type $FPrg^{V,A}$. Such logics define *equivalent transformations of programs*.

Another conventional program logic is *Floyd–Hoare logic*, which is based on assertions of the form $\{b_1\}s\{b_2\}$. Semantics of such assertions can be represented by Floyd–Hoare composition FH: $Pr^{V,A}\times FPrg^{V,A}\times Pr^{V,A} \xrightarrow{t} Pr^{V,A}$. We define this composition under assumption that predicates and functions can be partial [7]. Then

$$FH(p, prg, q)(d)=\begin{cases} T, & \text{if } q(prg(d))\downarrow = T \text{ or } p(d)\downarrow = F, \\ F, & \text{if } p(d)\downarrow = T \text{ and } q(prg(d))\downarrow = F, \\ & \text{undefined in other cases.} \end{cases}$$

Extending the algebra $APQ(V, A)$ with *FH* composition, unary parametric compositions of renomination $R\frac{\bar{v}}{\bar{x}}$, existential quantification $\exists x$ (to be defined in the next sections), and a composition of equality = we obtain a *three-sorted algebra*

$$APQFH(V, A) = < Pr^{V,A}, Fn^{V,A}, FPrg^{V,A};\ \vee, \neg, R\tfrac{\bar{v}}{\bar{x}},\ S_F^{\bar{v}},\ S_P^{\bar{v}},\ 'x, \exists x, =, AS^x, \bullet, IF, WH, FH >$$

presented in Fig. 1.

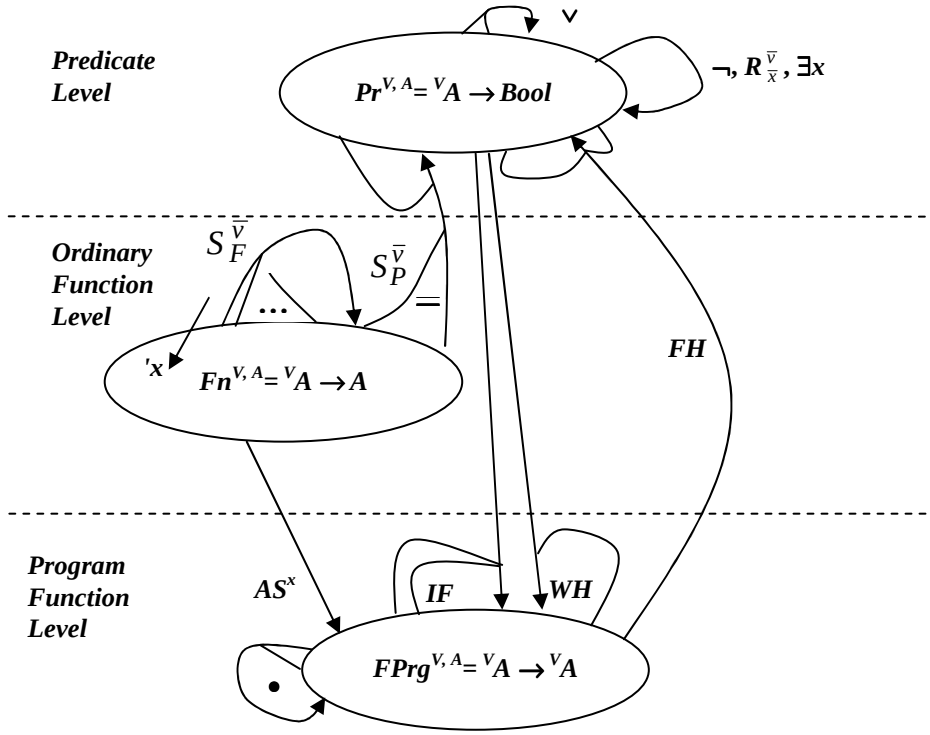

Figure 1. Three-sorted algebra of quasiary predicates, ordinary and program functions.

2.8 General program algebras as a semantic base of program logics

The class of algebras $APQFH(V, A)$ forms the semantic base for quite powerful Floyd–Hoare-like logics of quasiary mappings. It is important to admit that by restricting this algebra on one and two carriers we obtain respectively the following two algebras:

- *propositional algebra* $AP(V, A) = <Pr^{V,A};\ \vee, \neg >$;
- *first-order algebra of quasiary predicates and ordinary functions*

$$AQFO(V, A) = <Pr^{V,A}, Fn^{V,A}; \vee, \neg, R\tfrac{\bar{v}}{\bar{x}},\ S_F^{\bar{v}},\ S_P^{\bar{v}}, 'x, \exists x, = >.$$

Based on classes of such algebras various propositional, first-order, and program CNL can be defined (see the next sections). We can go further and define *modal and temporal CNL* [8].

All logics are constructed in a semantic-syntactic style: first, semantic component of a logic is defined as a class of certain algebras, then syntactic component (a logic language) is described, and, at last, interpretational component is specified.

2.9 Special features of quasiary predicates and functions

Before continuing with formal definitions it is desirable to give an intuitive understanding of logics of quasiary predicates and to compare them with classical logic of *n*-ary predicates.

We identify the following properties of quasiary predicates:

– *partiality* of predicates;
– *unrestricted* (possibly infinite) *arity* of predicates;
– *sensitivity* of predicates to unassigned variables (sensitivity to partiality of data).

These features complicate investigation of logics of quasiary predicates comparing with classical logic and violate some laws of classical logic.

In particular, partiality of predicates *violates Modus Ponens* [9]. Indeed, let *und* be interpreted as a nowhere defined predicate and Φ be interpreted as a refutable predicate. We treat validity as irrefutability, therefore *und* and $und \rightarrow \Phi$ are valid (irrefutable), but Φ is refutable.

Unrestricted arity of quasiary predicates *violates the property* that a sentence (a closed formula) has a *constant value* in a fixed interpretation because a predicate, obtained by formula interpretation, can depend upon variables that do not occur in that formula.

Sensitivity also affects the logic laws. Let us illustrate this with a simple example. Define a parametric predicate $\varepsilon z \in Pr^{V,A}$ ($z \in V$) by the following formulas: $\varepsilon z\,(d)\downarrow = F$ if z is assigned in d and $\varepsilon z\,(d)\downarrow = T$ if z is unassigned. This quasiary predicate is called *variable unassignment predicate*. It has different values depending whether a value of z is assigned or unassigned; thus, it is sensitive to unassigned variables. Consider the following phrase as one of intuitive interpretations of a predicate $\varepsilon food$: '*My cat is unhappy only in situations when he does not have any food*'. Thus, this phrase evaluates to *T* in situations (states) in which *food* is not assigned; when this variable is assigned then the phrase evaluates to *F*. The formula $(\forall food(\neg\varepsilon food)) \rightarrow \neg\varepsilon food$ may *fail* in situations (states) when my cat does not have any food. In Tarski's semantics of classical logic we have double totality: predicates are total and variable assignments (data) are total. Therefore in classical logic the formula $(\forall food(\neg\varepsilon food)) \rightarrow \neg\varepsilon food$ is always valid. Such sensitivity complicates logic calculus therefore more powerful instruments should be introduced. As such instruments we will use an infinite set of unessential variables and variable unassignment predicates (see the next sections).

Summing up, we would like to say that semantics of programs can be presented by terms of program algebras with compositions as operations in these algebras; program functions, ordinary functions, and predicates are quasiary mappings defined on nominative sets (nominative data); we define program logics directly on program algebras by extending their signatures with special "logical" compositions.

3 Hierarchy of Composition-Nominative Logics

Three kinds of logics can be constructed from program models over nominative sets (see Fig. 1):

1) *pure quasiary predicate logics based on algebras with one sort*: $Pr^{V,A}$;
2) *quasiary predicate-function logics based on algebras with two sorts*: $Pr^{V,A}$ and $Fn^{V,A}$;
3) *quasiary program logics based on algebras with three sorts*: $Pr^{V,A}$, $Fn^{V,A}$, and $FPrg^{V,A}$.

For logics of pure quasiary predicates (*pure* CNL) we identify renominative, quantifier, and quantifier-equational levels.

Renominative logics [8] are the most abstract among above-mentioned logics. The main new compositions for these logics are the compositions of renomination (renaming) of the form $R^{v_1,\ldots,v_n}_{x_1,\ldots,x_n} : Pr^{V,A} \xrightarrow{t} Pr^{V,A}$. Intuitively, given a quasiary predicate p and a nominative set d the value of $R^{v_1,\ldots,v_n}_{x_1,\ldots,x_n}(p)(d)$ is evaluated in the following way: first, a new nominative set d' is constructed from d by changing the values of the names $v_1,\ldots,v_n$ in d to the values of the names $x_1,\ldots,x_n$ respectively; then the predicate p is applied to d'. The obtained value (if it was evaluated) will be the result of $R^{v_1,\ldots,v_n}_{x_1,\ldots,x_n}(p)(d)$. For this composition we will also use a simplified notation $R^{\bar{v}}_{\bar{x}}$. The basic compositions of renominative logics are $\vee$, $\neg$, and $R^{\bar{v}}_{\bar{x}}$. Note, that renomination (primarily in syntactical aspects) is widely used in classical logic, lambda-calculus, and specification languages like Z-notation [10], B [11], TLA [12], RAISE [13], ASM [14] etc.

At the *quantifier* level, all basic values can be used to construct different nominative sets to which quasiary predicates can be applied. This allows one to introduce the compositions of quantification of the form $\exists x$ in style of Kleene's strong quantifiers. The basic compositions of logics of the quantifier level are $\vee$, $\neg$, $R^{\bar{v}}_{\bar{x}}$, and $\exists x$.

At the *quantifier-equational* level, new possibilities arise for equating and differentiating values with special 0-ary compositions of the form $=_{xy}$ called equality predicates. Basic compositions of logics of the quantifier-equational level are $\vee$, $\neg$, $R^{\bar{v}}_{\bar{x}}$, $\exists x$, and $=_{xy}$.

All specified logics (renominative, quantifier, and quantifier-equational) are based on algebras that have only one sort: a class of quasiary predicates.

For quasiary predicate-function logics we identify the function level and the function-equational level.

At the *function* level, we have extended capabilities of formation of new arguments for functions and predicates. In this case it is possible to introduce the superposition compositions $S_F^{\bar{v}}$ and $S_P^{\bar{v}}$ (see [8]), which formalize substitution of functions into function and predicate respectively. Also special 0-ary denomination parametric compositions $'x$ are introduced. Introduction of such functions allows one to model renomination compositions with the help of superpositions. The basic compositions of logics of the function level are $\vee$, $\neg$, $S_F^{\bar{v}}$, $S_P^{\bar{v}}$, $\exists x$, and $'x$.

At the function-equational level, a special equality composition = can be introduced additionally [8]. The basic compositions of logics of the function-equational level are $\vee$, $\neg$, $S_F^{\bar{v}}$, $S_P^{\bar{v}}$, $\exists x$, $'x$, and = . At this level different classes of first-order logics can be presented.

This means that two-sorted algebras (with sets of predicates and functions as sorts and above-mentioned compositions as operations) form a semantic base for first-order CNL.

To preserve properties of classical first-order logic in first-order CNL we should restrict the class ${}^{V}A \xrightarrow{p} Bool$ of quasiary predicates. Namely, we introduce a class of equitone predicates and its different variations such as maxitotal equitone predicates, equicompatible predicates, etc. [8]. A predicate p: ${}^{V}A \xrightarrow{p} Bool$ is called *equitone* if for every d, $d' \in {}^{V}A$ such that $d \subseteq d'$ from $p(d)\downarrow = b$ follows that $p(d')\downarrow = b$; if an equitone predicate p is defined on all elements of A^V then p is said to be *maxitotal equitone*; if a predicate p is a restriction of some equitone predicate then p is *equicompatible* predicate. Here A^V denotes all total mappings from V to A (total assignments). Logics based on maxitotal equitone, equitone, and equicompatible predicates are the "closest" generalizations of the classical first-order logic that preserve its main properties. These logics are called *neoclassical logics* [8].

The level of *program logics* is quite rich. Investigation of such logics is a special challenge; here we only mention *Floyd-Hoare-type logic* based on a monotone Floyd-Hoare composition [7].

In the rest of the paper we consider a *first-order composition-nominative pure quasiary predicate logic* denoted L^Q. Such logic plays a central role in logic hierarchy and is based on algebras $AQ(V, A) = < Pr^{V,A}, \vee, \neg, R_{\bar{x}}^{\bar{v}}, \exists x >$ (for various A). We will construct a sequent calculus for this logic and prove its soundness and completeness.

4 Formal Definitions of First-Order Composition-Nominative Pure Quasiary Predicate Logic

To define the logic L^Q we have to specify its semantic, syntactic, and interpretational components [15, 16].

4.1 Semantic component

Let V be a *set of names*. According to tradition, names from V are also called *variables*. Let A be a set of basic values. Given V and A, the class ${}^{V}A$ of nominative sets is defined as the class of all partial mappings from V to A, thus, ${}^{V}A = V \xrightarrow{p} A$. Informally speaking, nominative sets represent states of variables.

Though nominative sets are defined as mappings, we follow mathematical traditions and also use set-like notation for these objects. In particular, the notation $d = [v_i \mapsto a_i \mid i \in I]$ describes a nominative set d; the notation $v_i \mapsto a_i \in_n d$ means that $d(v_i)$ is defined and its value is a_i ($d(v_i)\downarrow = a_i$). The main operation for nominative sets is a total unary parametric *renomination* $\mathrm{r}_{x_1,\dots,x_n}^{v_1,\dots,v_n} : {}^{V}A \xrightarrow{t} {}^{V}A$ where $v_1, \dots, v_n, x_1, \dots, x_n \in V$, $v_1, \dots, v_n$ are distinct names, $n \geq 0$, which is defined by the following formula: $\mathrm{r}_{x_1,\dots,x_n}^{v_1,\dots,v_n}(d) = [v \mapsto a \in_n d \mid v \notin \{v_1, \dots, v_n\}] \cup [v_i \mapsto d(x_i) \mid d(x_i)\downarrow, i \in \{1,\dots,n\}]$. Intuitively, given d this operation yields a new nominative set changing the values of $v_1, \dots, v_n$ on the values of $x_1, \dots, x_n$ respectively. *The set of assigned names* in d is defined by the formula $asn(d) = \{v \in V \mid v \mapsto a \in_n d$ for some $a \in A\}$.

Let $Bool = \{F, T\}$ be a set of Boolean values. Let $Pr^{V,A} = {}^{V}A \xrightarrow{p} Bool$ be a set of all partial predicates over ${}^{V}A$. Such predicates are called *partial quasiary predicates*. The term 'partial' is usually omitted.

For $p \in Pr^{V,A}$, $d \in {}^{V}A$, $v \in V$, $a \in A$ we write:

- $p(d)\downarrow$ to denote that p is defined on a nominative set d;
- $p(d)\downarrow = b$ to denote that p is defined on d with a Boolean value b;
- $p(d)\uparrow$ to denote that p is undefined on d;

– $d(v)\downarrow$ to denote that a component with a name v is present in d;
– $d(v)\downarrow = a$ to denote that $v \mapsto a \in_n d$;
– $d(v)\uparrow$ to denote that the value of the name v is undefined in d.

The truth and falsity domains of p are respectively $T(p) = \{d \in {}^{V}A \mid p(d)\downarrow = T\}$ and $F(p) = \{d \in {}^{V}A \mid p(d)\downarrow = F\}$. A predicate p is *irrefutable*, or *partially valid*, if $F(p) = \varnothing$.

types Operations over $Pr^{V,A}$ are called *compositions*. For L^Q the set $C(V)$ of compositions is $\{\vee, \neg, R^{\bar{v}}_{\bar{x}}, \exists x\}$. Compositions have the following: $\vee$: $Pr^{V,A} \times Pr^{V,A} \xrightarrow{t} Pr^{V,A}$; $\neg$, $R^{v_1,\ldots,v_n}_{x_1,\ldots,x_n}$, $\exists x$: $Pr^{V,A} \xrightarrow{t} Pr^{V,A}$ and are defined by the following formulas ($p, q \in Pr^{V,A}$):

– $T(p \vee q) = T(p) \cup T(q)$; $F(p \vee q) = F(p) \cap F(q)$;
– $T(\neg p) = F(p)$; $F(\neg p) = T(p)$;
– $T(R^{\bar{v}}_{\bar{x}}(p)) = r^{\bar{v}}_{\bar{x}}(T(p))$; $F(R^{\bar{v}}_{\bar{x}}(p)) = r^{\bar{v}}_{\bar{x}}(F(p))$;
– $T(\exists xp) = \{d \in {}^{V}A \mid p(d\nabla x \mapsto a) = T \text{ for some } a \in A\}$; $F(\exists xp) = \{d \in {}^{V}A \mid p(d\nabla x \mapsto a) = F \text{ for every } a \in A\}$.

Here $d\nabla x \mapsto a = [v \mapsto c \in_n d \mid v \neq x] \cup [x \mapsto a]$.

Note that parametric compositions of existential quantification and renomination can also represent classes of compositions. Thus, notation $\exists x$ can represent one composition, when x is fixed, or a class $\{\exists x \mid x \in V\}$ of such compositions for various names. Also note that we treat a parameter $\begin{smallmatrix} v_1,\ldots,v_n \\ x_1,\ldots,x_n \end{smallmatrix}$ of renomination composition as a total mapping from $\{v_1, \ldots, v_n\}$ into $\{x_1, \ldots, x_n\}$ thus parameters obtained by pairs permutations are identical.

A pair $AQ(V, A) = \langle Pr^{V,A}, C(V)\rangle$ is called *a first-order algebra of quasiary predicates*. Such algebras form a semantic base for the constructed first-order composition-nominative pure quasiary predicate logic L^Q. Let us now proceed with syntactic and interpretational components of this logic.

4.2 Syntactic component

A syntactic component specifies the language of L^Q. Let $Cs(V)$ be *a set of composition symbols* that represent compositions in algebras defined above, $Cs(V) = \{\vee, \neg, R^{\bar{v}}_{\bar{x}}, \exists x\}$. For simplicity, we use the same notation for symbols of compositions and compositions themselves.

Let Ps be a set of *predicate symbols*. A triple $\Sigma^Q = (V, Cs(V), Ps)$ is a *signature* of a language of L^Q. Given a language signature Σ^Q, we inductively define the language of L^Q – *the set of formulas* $Fr(\Sigma^Q)$:

1) if $P \in Ps$ then $P \in Fr(\Sigma^Q)$;
2) if $\Phi, \Psi \in Fr(\Sigma^Q)$ then $(\Phi \vee \Psi) \in Fr(\Sigma^Q)$;
3) if $\Phi \in Fr(\Sigma^Q)$ then $\neg\Phi \in Fr(\Sigma^Q)$;
4) if $\Phi \in Fr(\Sigma^Q)$, $v_1, \ldots, v_n$, $x_1, \ldots, x_n \in V$, $v_1, \ldots, v_n$ are distinct names, $n \geq 0$, then $R^{v_1,\ldots,v_n}_{x_1,\ldots,x_n}\Phi \in Fr(\Sigma^Q)$;
5) if $\Phi \in Fr(\Sigma^Q)$, $x \in V$ then $\exists x\Phi \in Fr(\Sigma^Q)$.

4.3 Interpretational component

Given Σ^Q and a set A we can define an algebra of quasiary predicates $AQ(V, A) = \langle Pr^{V,A}, C(V)\rangle$. Composition symbols have fixed interpretation, but we additionally need interpretation $I^{Ps}: Ps \xrightarrow{t} Pr^{V,A}$ of predicate symbols to obtain a language interpretation. A corresponding tuple $J = (\Sigma^Q, I^{Ps})$ is called *an L^Q-interpretation*.

Given a formula Φ and an L^Q-interpretation J we can speak of an *interpretation of* Φ in J. It is denoted by Φ_J.

For the logic L^Q derived compositions (such as conjunction &, universal quantification $\forall x$, etc.) are defined in a traditional way.

Formulas and interpretations in L^Q are called L^Q *-formulas* and L^Q *-interpretations* respectively. Usually the prefix L^Q is omitted. A formula Φ is called *valid in interpretation J* if there is no nominative set $d \in {}^V A$ such that $\Phi_J(d)\downarrow = F$. This is denoted $J \models \Phi$, which means that Φ is not refutable in *J*. A formula Φ is called *valid* if $J \models \Phi$ for every interpretation *J*. We shall denote this $\models_{L^Q} \Phi$, or just $\models \Phi$ if the logic in hand is understood from the context.

4.4 Extensions of L^Q

The logic L^Q being a rather powerful logic still is not expressible enough to represent transformations required for proving its completeness. Therefore we introduce its two extensions: L^U *– a logic with unessential variables*, and L^U_ε – a logic with unessential variables and a *parametric variable unassignment predicate* εz which checks whether a variable *z* is unassigned in a given nominative set.

To define L^U we should specify its semantic, syntactic, and interpretational components.

Let *U* be an infinite set of variables such that $V \cap U = \varnothing$. Variables from *U* are called *unessential variables* (analogs of fresh variables in classical logic) that should not affect the formula meanings [15]. Algebras $AQ(V \cup U, A) = < Pr^{V \cup U, A}, C(V) >$ (for different *A*) form a semantic base for L^U. A syntactic component is specified by the set of formulas $Fr(\Sigma^U)$ where $\Sigma^U = (V \cup U, Cs(V \cup U), Ps)$ is the *signature* of L^U. An interpretational component of L^U restricts the class of L^U-interpretations in such a way that interpretations of predicate symbols are neither sensitive to the values of the component with an unessential variable *u* in nominative sets, nor to presence of such components. Formally, a variable $u \in U$ is *unessential in an interpretation* I^{Ps} if $I^{Ps}(P)(d) = I^{Ps}(P)(d\nabla u \mapsto a)$ for all $P \in Ps$, $d \in {}^{V \cup U}A$, $a \in A$.

The logic L^U_ε is an extension of L^U by a null-ary parametric composition (predicate) εz ($z \in V \cup U$) defined by the formulas: $T(\varepsilon z_A) = \{d \mid d(z)\uparrow\} = \{d \in {}^V A \mid z \notin asn(d)\}$ and $F(\varepsilon z_A) = \{d \mid d(z)\downarrow\} = \{d \in {}^V A \mid z \in asn(d)\}$. Thus, for this logic the set of compositions is equal to $\{\vee, \neg, R^{\bar{v}}_{\bar{x}}, \exists x, \varepsilon z\}$. Algebras of the form $AQE(V \cup U, A) = < Pr^{V \cup U, A}, \vee, \neg, R^{\bar{v}}_{\bar{x}}, \exists x, \varepsilon z >$ constitute a semantic base for L^U_ε. A syntactic component is specified by the set of formulas $Fr(\Sigma^U_\varepsilon)$ where $\Sigma^U_\varepsilon = (V \cup U, \{\vee, \neg, R^{\bar{v}}_{\bar{x}}, \exists x, \varepsilon z\}, Ps)$ is the signature of L^U. An interpretational component of L^U_ε is defined in the same way as for L^U.

Predicates εz specify cases when *z* is assigned or unassigned. This property can be used for quantifier elimination that is semantically supported for every algebra $AQE(V \cup U, A)$ by the following statement:

$$T(R^{\bar{u},x}_{\bar{v},y}(P)) \cap F(\varepsilon y) \subseteq T(R^{\bar{u}}_{\bar{v}}(\exists x P)) \text{ and } F(R^{\bar{u}}_{\bar{v}}(\exists x P)) \cap F(\varepsilon y) \subseteq F(R^{\bar{u},x}_{\bar{v},y}(P)).$$

Proof. Let $d \in T(R^{\bar{u},x}_{\bar{v},y}(P)) \cap F(\varepsilon y)$, then $d(y)\downarrow$ and $R^{\bar{u},x}_{\bar{v},y}(P)(d) = T$, therefore $d(y)\downarrow a$ for some $a \in A$ and $P(d\nabla \bar{u} \mapsto d(\bar{v})\nabla x \mapsto d(y)) = T$. Hence, $P(d\nabla \bar{u} \mapsto d(\bar{v})\nabla x \mapsto a) = T$ for some $a \in A$. Thus, $(\exists xP)(r^{\bar{u}}_{\bar{v}}(d)) = T$, therefore $R^{\bar{u}}_{\bar{v}}(\exists xP)(d) = T$; this means that $d \in T(R^{\bar{u}}_{\bar{v}}(\exists xP))$. So, $T(R^{\bar{u},x}_{\bar{v},y}(P)) \cap F(\varepsilon y) \subseteq T(R^{\bar{u}}_{\bar{v}}(\exists xP))$.

Let $d \in F(R^{\bar{u}}_{\bar{v}}(\exists xP)) \cap F(\varepsilon y)$, then $d(y)\downarrow$ and $R^{\bar{u}}_{\bar{v}}(\exists xP)(d) = F$. From this $(\exists xP)(d\nabla \bar{u} \mapsto d(\bar{v})) = F$, therefore $P(d\nabla \bar{u} \mapsto d(\bar{v})\nabla x \mapsto b) = F$ for all $b \in A$. Since $d(y)\downarrow$ we have $d(y)\downarrow a$ for some $a \in A$, then $P(d\nabla \bar{u} \mapsto d(\bar{v})\nabla x \mapsto d(y)) = F$. From this $R^{\bar{u},x}_{\bar{v},y}(P)(d) = F$, which gives $d \in F(R^{\bar{u},x}_{\bar{v},y}(P))$. So, $F(R^{\bar{u}}_{\bar{v}}(\exists xP)) \cap F(\varepsilon y) \subseteq F(R^{\bar{u},x}_{\bar{v},y}(P))$.

As a special case we get $T(R^x_y(P)) \cap F(\varepsilon y) \subseteq T(\exists xP)$ та $F(\exists xP) \cap F(\varepsilon y) \subseteq F(R^x_y(P))$.

5 Consequence Relation for Sets of Formulas

Consequence relation is defined in the same way for all logics under consideration. So, we present its definition only for a logic L^U_ε.

Let $\Gamma \subseteq Fr(\Sigma^U_\varepsilon)$ and $\Delta \subseteq Fr(\Sigma^U_\varepsilon)$ be sets of formulas. Δ is a *logical consequence* of Γ in an interpretation *J* (denoted by $\Gamma_J \models \Delta$), if $\bigcap_{\Phi \in \Gamma} T(\Phi_J) \cap \bigcap_{\Psi \in \Delta} F(\Psi_J) = \varnothing$. Δ is a logical consequence of Γ (denoted by $\Gamma \models \Delta$), if $\Gamma_J \models \Delta$ in every inter-

pretation J. The set of names (variables) that occur in Γ is denoted by $nm(\Gamma)$, this notation is extended for a case of several sets or formulas, say, $nm(\Gamma, \Delta, R^{\bar{u}}_{\bar{v}}(\exists x\Phi))$.

General properties of the consequence relation are the following ($\Gamma, \Delta, \Sigma, \Lambda \subseteq Fr(\Sigma^U_\varepsilon)$):

C) Let $\Gamma \cap \Delta \neq \varnothing$, then $\Gamma \models \Delta$.
U) Let $\Gamma \subseteq \Lambda$ and $\Delta \subseteq \Sigma$, then $\Gamma \models \Delta \Rightarrow \Lambda \models \Sigma$.

General properties with their duals (co-rules) are the following ($\Gamma, \Delta, \Sigma \subseteq Fr(\Sigma^U_\varepsilon)$, $\Phi,\ \Psi \in Fr(\Sigma^U_\varepsilon)$):

$\vee_{\vdash}$) $\Phi\vee\Psi, \Gamma \models \Delta \Leftrightarrow \Phi, \Gamma \models \Delta$ та $\Psi, \Gamma \models \Delta$;	$\vee_{\dashv}$) $\Gamma \models \Delta, \Phi\vee\Psi \Leftrightarrow \Gamma \models \Delta, \Phi, \Psi$;
RT$_{\vdash}$) $R^{z,\bar{v}}_{z,\bar{x}}(\Phi), \Gamma \models \Delta \Leftrightarrow R^{\bar{v}}_{\bar{x}}(\Phi), \Gamma \models \Delta$;	RT$_{\dashv}$) $\Gamma \models \Delta, R^{z,\bar{v}}_{z,\bar{x}}(\Phi) \Leftrightarrow \Gamma \models \Delta, R^{\bar{v}}_{\bar{x}}(\Phi)$;
ΦN$_{\vdash}$) $R^{y,\bar{v}}_{z,\bar{x}}(\Phi), \Gamma \models \Delta \Leftrightarrow R^{\bar{v}}_{\bar{x}}(\Phi), \Gamma \models \Delta$, if $y \in U \setminus nm(\Phi)$;	ΦN$_{\dashv}$) $\Gamma \models \Delta, R^{y,\bar{v}}_{z,\bar{x}}(\Phi) \Leftrightarrow \Gamma \models \Delta, R^{\bar{v}}_{\bar{x}}(\Phi)$, if $y \in U \setminus nm(\Phi)$;
RR$_{\vdash}$) $R^{\bar{v}}_{\bar{x}}(R^{\bar{w}}_{\bar{y}}(\Phi)), \Gamma \models \Delta \Leftrightarrow R^{\bar{v}}_{\bar{x}} \circ^{\bar{w}}_{\bar{y}}(\Phi), \Gamma \models \Delta$;	RR$_{\dashv}$) $\Gamma \models \Delta, R^{\bar{v}}_{\bar{x}}(R^{\bar{w}}_{\bar{y}}(\Phi)) \Leftrightarrow \Gamma \models \Delta, R^{\bar{v}}_{\bar{x}} \circ^{\bar{w}}_{\bar{y}}(\Phi)$;
R$\neg_{\vdash}$) $R^{\bar{v}}_{\bar{x}}(\neg\Phi), \Gamma \models \Delta \Leftrightarrow \neg R^{\bar{v}}_{\bar{x}}(\Phi), \Gamma \models \Delta$;	R$\neg_{\dashv}$) $\Gamma \models \Delta, R^{\bar{v}}_{\bar{x}}(\neg\Phi) \Leftrightarrow \Gamma \models \Delta, \neg R^{\bar{v}}_{\bar{x}}(\Phi)$;
R$\vee_{\vdash}$) $R^{\bar{v}}_{\bar{x}}(\Phi\vee\Psi), \Gamma \models \Delta \Leftrightarrow R^{\bar{v}}_{\bar{x}}(\Phi)\vee R^{\bar{v}}_{\bar{x}}(\Psi), \Gamma \models \Delta$;	R$\vee_{\dashv}$) $\Gamma \models \Delta, R^{\bar{v}}_{\bar{x}}(\Phi\vee\Psi) \Leftrightarrow \Gamma \models \Delta, R^{\bar{v}}_{\bar{x}}(\Phi)\vee R^{\bar{v}}_{\bar{x}}(\Psi)$;
R$\exists$R$_{\vdash}$) $R^{\bar{u},x}_{\bar{v},y}(\exists x\Phi), \Gamma \models \Delta \Leftrightarrow R^{\bar{u}}_{\bar{v}}(\exists x\Phi), \Gamma \models \Delta$;	R$\exists$R$_{\dashv}$) $\Gamma \models \Delta, R^{\bar{u},x}_{\bar{v},y}(\exists x\Phi) \Leftrightarrow \Gamma \models \Delta, R^{\bar{u}}_{\bar{v}}(\exists x\Phi)$;
R$\exists$p$_{\vdash}$) $R^{x}_{y}(\exists x\Phi), \Gamma \models \Delta \Leftrightarrow \exists x\Phi, \models \Delta$;	R$\exists$p$_{\dashv}$) $\Gamma \models \Delta, R^{x}_{y}(\exists x\Phi) \Leftrightarrow \models \Delta, \exists x\Phi$.

Properties related to elimination of quantifiers are the following:

$\exists$R$_{\vdash}$) $R^{\bar{u}}_{\bar{v}}(\exists x\Phi), \Gamma \models \Delta \Leftrightarrow R^{\bar{u},x}_{\bar{v},z}(\Phi), \Gamma \models \Delta, \varepsilon z$, if $z \in U \setminus nm(\Gamma, \Delta, R^{\bar{u}}_{\bar{v}}(\exists x\Phi))$;

$\exists_{\vdash}$) $\exists x\Phi, \Gamma \models \Delta \Leftrightarrow R^{x}_{z}(\Phi), \Gamma \models \Delta, \varepsilon z$, if $z \in U \setminus nm(\Gamma, \Delta, \exists x\Phi)$;

$\exists$Rf$_{\dashv}$) $\Gamma \models \Delta, R^{\bar{u}}_{\bar{v}}(\exists x\Phi) \Leftrightarrow \Gamma \models \Delta, R^{\bar{u}}_{\bar{v}}(\exists x\Phi), R^{\bar{u},x}_{\bar{v},z}(\Phi), \varepsilon z$, if $z \in U \setminus nm(\Gamma, \Delta, R^{\bar{u}}_{\bar{v}}(\exists x\Phi))$;

$\exists$f$_{\dashv}$) $\Gamma \models \Delta, \exists x\Phi \Leftrightarrow \Gamma \models \Delta, \exists x\Phi, R^{x}_{z}(\Phi), \varepsilon z$, if $z \in U \setminus nm(\Gamma, \Delta, \exists x\Phi)$;

$\exists$Rv$_{\dashv}$) $\Gamma \models \Delta, R^{\bar{u}}_{\bar{v}}(\exists x\Phi), \varepsilon y \Leftrightarrow \Gamma \models \Delta, R^{\bar{u}}_{\bar{v}}(\exists x\Phi), R^{\bar{u},x}_{\bar{v},y}(\Phi), \varepsilon y$.

$\exists$v$_{\dashv}$) $\Gamma \models \Delta, \exists x\Phi, \varepsilon y \Leftrightarrow \Gamma \models \Delta, \exists x\Phi, R^{x}_{y}(\Phi), \varepsilon y$.

$\exists$Rd$_{\dashv}$) $\Gamma \models \Delta, R^{\bar{u}}_{\bar{v}}(\exists x\Phi) \Leftrightarrow \varepsilon y, \Gamma \models \Delta, R^{\bar{u}}_{\bar{v}}(\exists x\Phi)$ and $\Gamma \models \Delta, R^{\bar{u}}_{\bar{v}}(\exists x\Phi), R^{\bar{u},x}_{\bar{v},y}(\Phi), \varepsilon y$;

$\exists$d$_{\dashv}$) $\Gamma \models \Delta, \exists x\Phi \Leftrightarrow \varepsilon y, \Gamma \models \Delta, \exists x\Phi$ and $\Gamma \models \Delta, \exists x\Phi, R^{x}_{y}(\Phi), \varepsilon y$.

Let us prove, for example, the property $\exists$R$_{\vdash}$ ($\exists_{\vdash}$ is its special case) and the property $\exists$Rd$_{\dashv}$ ($\exists$d$_{\dashv}$ is its special case) considering an arbitrary interpretation J.

Property $\exists$R$_{\vdash}$.

$\Rightarrow$. If $R^{\bar{u}}_{\bar{v}}(\exists x\Phi), \Gamma \models \Delta$ then $T(\Gamma_J) \cap T(R^{\bar{u}}_{\bar{v}}(\exists x\Phi)_J) \cap F(\Delta_J) = \varnothing$. Since $T(R^{\bar{u},x}_{\bar{v},z}(\Phi)_J) \cap F(\varepsilon z_J) \subseteq T(R^{\bar{u}}_{\bar{v}}(\exists x\Phi)_J)$, we have $T(\Gamma_J) \cap F(\Delta_J) \cap T(R^{\bar{u},x}_{\bar{v},z}(\Phi)_J) \cap F(\varepsilon z_J) = \varnothing$. So, $R^{\bar{u},x}_{\bar{v},z}(\Phi), \Gamma \models \Delta, \varepsilon z$.

$\Leftarrow$. Let $R^{\bar{u},x}_{\bar{v},z}(\Phi), \Gamma \models \Delta, \varepsilon z$, then $T(\Gamma_A) \cap T(R^{\bar{u},x}_{\bar{v},z}(\Phi)_J) \cap F(\Delta_J) \cap F(\varepsilon z_J) = \varnothing$. If we demonstrate that $T(\Gamma_A) \cap T(R^{\bar{u}}_{\bar{v}}(\exists x\Phi)_J) \cap F(\Delta_J) = \varnothing$ then we obtain $R^{\bar{u}}_{\bar{v}}(\exists x\Phi), \Gamma \models \Delta$.

Assume that $T(\Gamma_J) \cap T(R^{\bar{u},x}_{\bar{v},z}(\Phi)_J) \cap F(\Delta_J) \cap F(\varepsilon z_J) = \varnothing$ and there exists d such that $d \in T(\Gamma_J) \cap T(R^{\bar{u}}_{\bar{v}}(\exists x\Phi)_J) \cap F(\Delta_J)$. In this case $d \in T(R^{\bar{u}}_{\bar{v}}(\exists x\Phi)_J))$, $d \in T(\Gamma_J)$ and $d \in F(\Delta_J)$. By $d \in T(R^{\bar{u}}_{\bar{v}}(\exists x\Phi)_J)$ we have $d\nabla\bar{u} \mapsto d(\bar{v}) \in T(\exists x\Phi_J)$; this means that $d\nabla\bar{u} \mapsto d(\bar{v})\nabla x \mapsto a \in T(\Phi_J)$ for some $a \in A$. Since $z \in U \setminus nm(\Gamma, \Delta, R^{\bar{u}}_{\bar{v}}(\exists x\Phi))$ we have $d\nabla\bar{u} \mapsto d(\bar{v})\nabla x \mapsto a\nabla z \mapsto a \in T(\Phi_J)$, $d\nabla z \mapsto a \in T(\Gamma_J)$, $d\nabla z \mapsto a \in F(\Delta_J)$. From this follows that $d\nabla z \mapsto a \in T(R^{\bar{u},x}_{\bar{v},z}(\Phi)_J)$; by definition of εz we have $d\nabla z \mapsto a \in F(\varepsilon z_J)$, therefore $d\nabla z \mapsto a \in T(\Gamma_J) \cap T(R^{\bar{u},x}_{\bar{v},z}(\Phi)_J) \cap F(\Delta_J) \cap F(\varepsilon z_J)$. This contradicts to the assumption that $T(\Gamma_J) \cap T(R^{\bar{u},x}_{\bar{v},z}(\Phi)_J) \cap F(\Delta_J) \cap F(\varepsilon z_J) = \varnothing$.

Property $\exists$Rd$_{\dashv}$.

$\Rightarrow$. If $\Gamma \models \Delta, R^{\bar{u}}_{\bar{v}}(\exists x\Phi)$, then $\Gamma \models \Delta, R^{\bar{u}}_{\bar{v}}(\exists x\Phi), R^{\bar{u},x}_{\bar{v},y}(\Phi), \varepsilon y$ and $\varepsilon y, \Gamma \models \Delta, R^{\bar{u}}_{\bar{v}}(\exists x\Phi)$ by the general property U).

$\Leftarrow$. Assume that $\varepsilon y, \Gamma \models \Delta, R^{\bar{u}}_{\bar{v}}(\exists x\Phi)$ and $\Gamma \models \Delta, R^{\bar{u}}_{\bar{v}}(\exists x\Phi), R^{\bar{u},x}_{\bar{v},y}(\Phi), \varepsilon y$, but $\Gamma \not\models \Delta, R^{\bar{u}}_{\bar{v}}(\exists x\Phi)$. Then we have $T(\Gamma_J) \cap F(\Delta_J) \cap F(R^{\bar{u}}_{\bar{v}}(\exists x\Phi)_J) \neq \varnothing$; this implies that there exists d such that $d \in T(\Gamma_J) \cap F(\Delta_J) \cap F(R^{\bar{u}}_{\bar{v}}(\exists x\Phi)_J)$.

Two cases are possible: $d(y)\uparrow$ and $d(y)\downarrow$. If $d(y)\uparrow$ then $d \in T(\varepsilon y_J)$; from this $d \in T(\varepsilon y_J) \cap T(\Gamma_J) \cap F(\Delta_J) \cap F(R^{\bar{u}}_{\bar{v}}(\exists x\Phi)_J)$ that contradicts to $\varepsilon y, \Gamma \models \Delta, R^{\bar{u}}_{\bar{v}}(\exists x\Phi)$. If $d(y)\downarrow$ then $d \in F(\varepsilon y_J)$. Let $d(y) = a$. By $d \in F(R^{\bar{u}}_{\bar{v}}(\exists x\Phi)_J)$ we have $d\nabla\bar{u} \mapsto d(\bar{v}) \in F(\exists x\Phi_J)$. From this follows $d\nabla\bar{u} \mapsto d(\bar{v})\nabla x \mapsto b \in F(\Phi_J)$ for every $b \in A$, in particular, $d\nabla\bar{u} \mapsto d(\bar{v})\nabla x \mapsto a \in F(\Phi_J)$, therefore $d\nabla\bar{u} \mapsto d(\bar{v})\nabla x \mapsto d(y) \in F(\Phi_J)$, thus, $d \in F(R^{\bar{u},x}_{\bar{v},y}(\Phi)_J)$. So, $d \in T(\Gamma_J) \cap F(\Delta_J) \cap$ $\cap F(R^{\bar{u}}_{\bar{v}}(\exists x\Phi)_J) \cap F(R^{\bar{u},x}_{\bar{v},y}(\Phi)_J) \cap F(\varepsilon y_J)$ that contradicts $\Gamma \models \Delta, R^{\bar{u}}_{\bar{v}}(\exists x\Phi), R^{\bar{u},x}_{\bar{v},y}(\Phi), \varepsilon y$.

6 The sequent calculus for L^Q

For the logic L^Q we build a *calculus of sequent type*. Sequents are interpreted as sets of *labeled (signed) formulas* marked by one of two symbols $\vdash$ or $\dashv$. Such sequents Σ are also denoted by $\vdash\Gamma\dashv\Delta$, where all formulas of Γ are labeled by the symbol $\vdash$ (such formulas are called *T*-formulas), of Δ – by the symbol $\dashv$ (*F*-formulas). This notation for sequents is similar to notations used in tableau calculi.

Semantic properties of relation $\models$ have their syntactic analogues – sequent rules. These rules are the following.

Sequent rules for propositional compositions:

$$\vdash\neg\ \frac{\dashv\Phi, \Sigma}{\vdash\neg\Phi, \Sigma}; \qquad \dashv\neg\ \frac{\vdash\Phi, \Sigma}{\dashv\neg\Phi, \Sigma}.$$

$$\vdash\vee\ \frac{\vdash\Phi, \Sigma \quad \vdash\Psi, \Sigma}{\vdash\Phi\vee\Psi, \Sigma}; \qquad \dashv\vee\ \frac{\dashv\Phi, \dashv\Psi, \Sigma}{\dashv\Phi\vee\Psi, \Sigma}.$$

Sequent rules for renomination compositions:

$$\vdash\mathrm{RT}\ \frac{\vdash R^{\bar{v}}_{\bar{x}}(\Phi), \Sigma}{\vdash R^{z,\bar{v}}_{z,\bar{x}}(\Phi), \Sigma}; \qquad \dashv\mathrm{RT}\ \frac{\dashv R^{\bar{v}}_{\bar{x}}(\Phi), \Sigma}{\dashv R^{z,\bar{v}}_{z,\bar{x}}(\Phi), \Sigma}.$$

$$\vdash\Phi\mathrm{N}\ \frac{\vdash R^{\bar{v}}_{\bar{u}}(\Phi), \Sigma}{\vdash R^{y,\bar{v}}_{z,\bar{u}}(\Phi), \Sigma}, \text{ if } y \in U\backslash nm(\Phi); \qquad \dashv\Phi\mathrm{N}\ \frac{\dashv R^{\bar{v}}_{\bar{u}}(\Phi), \Sigma}{\dashv R^{y,\bar{v}}_{z,\bar{u}}(\Phi), \Sigma}, \text{ if } y \in U\backslash nm(\Phi);$$

$$\vdash\mathrm{R}\exists\mathrm{R}\ \frac{\vdash R^{\bar{u}}_{\bar{v}}(\exists x\Phi), \Sigma}{\vdash R^{\bar{u},x}_{\bar{v},y}(\exists x\Phi), \Sigma}; \qquad \dashv\mathrm{R}\exists\mathrm{R}\ \frac{\dashv R^{\bar{u}}_{\bar{v}}(\exists x\Phi), \Sigma}{\dashv R^{\bar{u},x}_{\bar{v},y}(\exists x\Phi), \Sigma}.$$

$$\vdash\mathrm{R}\exists\mathrm{p}\ \frac{\vdash\exists x\Phi, \Sigma}{\vdash R^{x}_{y}(\exists x\Phi), \Sigma}; \qquad \dashv\mathrm{R}\exists\mathrm{p}\ \frac{\dashv\exists x\Phi, \Sigma}{\dashv R^{x}_{y}(\exists x\Phi), \Sigma}.$$

$$\vdash\mathrm{RR}\ \frac{\vdash R^{\bar{v}}_{\bar{x}} \circ^{\bar{w}}_{\bar{y}}(\Phi), \Sigma}{\vdash R^{\bar{v}}_{\bar{x}}(R^{\bar{w}}_{\bar{y}}(\Phi)), \Sigma}; \qquad \dashv\mathrm{RR}\ \frac{\dashv R^{\bar{v}}_{\bar{x}} \circ^{\bar{w}}_{\bar{y}}(\Phi), \Sigma}{\dashv R^{\bar{v}}_{\bar{x}}(R^{\bar{w}}_{\bar{y}}(\Phi)), \Sigma}.$$

$$\vdash\mathrm{R}\neg\ \frac{\vdash\neg R^{\bar{v}}_{\bar{x}}(\Phi), \Sigma}{\vdash R^{\bar{v}}_{\bar{x}}(\neg\Phi), \Sigma}; \qquad \dashv\mathrm{R}\neg\ \frac{\dashv\neg R^{\bar{v}}_{\bar{x}}(\Phi), \Sigma}{\dashv R^{\bar{v}}_{\bar{x}}(\neg\Phi), \Sigma}.$$

$$\vdash\mathrm{R}\vee\ \frac{\vdash R^{\bar{v}}_{\bar{x}}(\Phi)\vee R^{\bar{v}}_{\bar{x}}(\Psi), \Sigma}{\vdash R^{\bar{v}}_{\bar{x}}(\Phi\vee\Psi), \Sigma}; \qquad \dashv\mathrm{R}\vee\ \frac{\dashv R^{\bar{v}}_{\bar{x}}(\Phi)\vee R^{\bar{v}}_{\bar{x}}(\Psi), \Sigma}{\dashv R^{\bar{v}}_{\bar{x}}(\Phi\vee\Psi), \Sigma}.$$

Sequent rules for quantification compositions:

$$\vdash\exists\ \frac{\vdash R^{x}_{z}(\Phi), \dashv\varepsilon z, \Sigma}{\vdash\exists x\Phi, \Sigma}, \text{ if } z \in U\backslash nm(\Sigma, \exists x\Phi). \qquad \vdash\exists\mathrm{R}\ \frac{\vdash R^{\bar{u},x}_{\bar{v},z}(\Phi), \dashv\varepsilon z, \Sigma}{\vdash R^{\bar{u}}_{\bar{v}}(\exists x\Phi), \Sigma}, \text{ if } z \in U\backslash nm(\Sigma, R^{\bar{u}}_{\bar{v}}(\exists x\Phi)).$$

$\dashv\exists$f $\dfrac{\dashv\exists x\Phi,\ \dashv R_z^x(\Phi),\ \dashv \varepsilon z, \Sigma}{\dashv\exists x\Phi, \Sigma}$, if $z\in U\setminus nm(\Sigma, \exists x\Phi)$. $\dashv\exists$Rf $\dfrac{\dashv R_{\bar v}^{\bar u}(\exists x\Phi),\ \dashv R_{\bar v, z}^{\bar u, x}(\Phi),\ \dashv \varepsilon z, \Sigma}{\dashv R_{\bar v}^{\bar u}(\exists x\Phi), \Sigma}$, if $z\in U\setminus nm(\Sigma,\ R_{\bar v}^{\bar u}(\exists x\Phi))$.

$\dashv\exists$v $\dfrac{\dashv\exists x\Phi,\ \dashv R_y^x(\Phi),\ \dashv \varepsilon y, \Sigma}{\dashv\exists x\Phi,\ \dashv \varepsilon y, \Sigma}$. $\dashv\exists$Rv $\dfrac{\dashv R_{\bar v}^{\bar u}(\exists x\Phi),\ \dashv R_{\bar v, y}^{\bar u, x}(\Phi),\ \dashv \varepsilon y, \Sigma}{\dashv R_{\bar v}^{\bar u}(\exists x\Phi),\ \dashv \varepsilon y, \Sigma}$.

$\dashv\exists$d $\dfrac{\vdash \varepsilon y,\ \dashv \exists x\Phi, \Sigma \qquad \dashv\exists x\Phi,\ \dashv R_y^x(\Phi),\ \dashv \varepsilon y, \Sigma}{\dashv\exists x\Phi, \Sigma}$; $\dashv\exists$Rd $\dfrac{\vdash \varepsilon y,\ \dashv R_{\bar v}^{\bar u}(\exists x\Phi), \Sigma \qquad \dashv R_{\bar v}^{\bar u}(\exists x\Phi),\ \dashv R_{\bar v, y}^{\bar u, x}(\Phi),\ \dashv \varepsilon y, \Sigma}{\dashv R_{\bar v}^{\bar u}(\exists x\Phi), \Sigma}$.

Additional condition for $\dashv\exists$f and $\dashv\exists$Rf: predicate symbols εz do not belong Σ. Additional condition for $\dashv\exists$d and $\dashv\exists$Rd: εy, εz do not belong to Σ but Σ has at least one symbol of the form εz.

Sequent calculus specified by the above written rules is denoted as *Q-calculus*.

To define derivability in *Q*-calculus we should first introduce the notion of closed sequent. Sequent Σ is *closed*, if there exists Φ such that $\vdash\Phi\in\Sigma$ and $\dashv\Phi\in\Sigma$. Consequently, if $\vdash\Gamma\dashv\Delta$ is closed then $\Gamma \models \Delta$. Closed sequents are *axioms* of *Q*-calculus.

We also need an additional condition of closed sequents which is called *unas-closeness*. Given a sequent $\vdash\Gamma\dashv\Delta$ we define $unas(\vdash\Gamma\dashv\Delta) = \{u\in V \mid \varepsilon u\in\Gamma\}$. It is assumed that for variables from $unas(\vdash\Gamma\dashv\Delta)$ values are not assigned (for counter models specified by a derivation tree [17]). Given $\vdash\Gamma\dashv\Delta$ and two formulas $R_{\bar x}^{\bar v}(\Phi)$ and $R_{\bar y}^{\bar s}(\Phi)$ we say that these formulas are *unas-equivalent* if formal expressions obtained by deleting from $\bar v_{\bar x}$ and $\bar s_{\bar y}$ variables from $unas(\vdash\Gamma\dashv\Delta)$ coincide (exact definition is given in [17]). A sequent $\vdash\Gamma\dashv\Delta$ is *unas-closed* if there exist two *unas*-equivalent formulas $R_{\bar x}^{\bar v}(\Phi)$ and $R_{\bar y}^{\bar s}(\Phi)$ such that $R_{\bar x}^{\bar v}(\Phi)\in\Gamma$ and $R_{\bar y}^{\bar s}(\Phi)\in\Delta$.

Derivation in the *Q*-calculus has the form of tree, the vertices of which are sequents. Such trees are called sequent trees. A sequent tree is *closed*, if every its leaf is a closed sequent. A sequent Σ is *derivable*, if there is a closed sequent tree with the root Σ. Sequent calculus is constructed in such a way that a sequent $\vdash\Gamma\dashv\Delta$ has a derivation if and only if $\Gamma \models \Delta$. The derivability of a sequent for formulas of L^Q is proved within L_ε^U.

During construction of a sequent tree the following cases are possible:

1. All sequents on the leaves of the sequent tree are closed; we have a finite closed tree.

2. Procedure is not completed; we have a finite or infinite unclosed tree. Such tree has at least one path all vertices of which are unclosed sequents. Such path is called unclosed.

Theorem (soundness). Let a sequent $\vdash\Gamma\dashv\Delta$ be derivable in *Q*-calculus. Then $\Gamma \models \Delta$.

A proof is based on the semantic properties of the consequence relation presented in the previous section.

Theorem (completeness). Let $\Gamma \models \Delta$. Then the sequent $\vdash\Gamma\dashv\Delta$ is derivable in *Q*-calculus.

A proof is based on the fact that a counter model for a sequent can be constructed if its derivation does not exist. A detailed proof is lengthy one and is omitted here.

Q-calculus is a new simplified version of *QG*-calculus presented in [17]. *QG*-calculus was constructed for a special consequence relation, but here we adopt a traditional definition of this relation.

Using the ideas presented in this paper we plan to construct in forthcoming papers calculi for composition-nominative logics of quantifier-equational, function, and function-equational levels (see section 3) and give detailed proofs of their soundness and completeness.

In our previous work we constructed *calculi for different neoclassical logics* and proved their *soundness and completeness* [8]. Similar results were also proved for some classes of composition-nominative modal and temporal logics [8].

The obtained results can be used in logics for program reasoning.

7 Conclusions

In the paper we have advocated the idea that logics for program reasoning should be based directly on formal program models. In this case program logics should reflect such program features as partiality, complex data structures, nondeterminism etc. Program-oriented logics developed in the paper are called composition-nominative program logics and are algebra-based logics constructed in a semantic-syntactic style on the methodological basis that is common with program-

ming; they can be considered as generalizations of traditional logics on classes of partial predicates that do not have fixed arity. Such predicates, called quasiary predicates, are defined over partial variable assignments (partial data). We have described the hierarchy of different logics of quasiary predicates. For the constructed logics some laws of classical logic fail because of partiality of predicates and data. We have constructed a sequent calculus for a first-order composition-nominative pure quasiary predicate logic which plays the central role in the logic hierarchy. We have proved soundness and completeness of this calculus. The obtained results can be generalized for a number of more powerful logics. The proposed methods can be useful for construction and investigation of logics for program reasoning.

Future work on the topic will include construction of sequent calculi for composition-nominative logics over hierarchic nominative data. Hierarchic data permit to represent such complex structures as lists, stacks, arrays, etc.; thus, such logics will be closer to program models with more rich data types. Also, prototypes of software systems for theorem proving in composition-nominative logics should be developed.

References

1. Handbook of Logic in Computer Science, S. Abramsky, Dov M. Gabbay, and T. S. E. Maibaum (eds.), in 5 volumes, Oxford Univ. Press, Oxford (1993–2001)
2. Handbook of Philosophical Logic, D.M. Gabbay, F. Guenthner (eds.), 2nd Edition, in 16 volumes, Springer (2001–2011)
3. Nikitchenko, N.(M.): A Composition Nominative Approach to Program Semantics. Technical Report IT–TR 1998-020, Technical University of Denmark, 103 p. (1998)
4. Nielson H.R., Nielson F.: Semantics with Applications: A Formal Introduction. John Wiley & Sons Inc (1992)
5. Winskel G.: The Formal Semantics of Programming Languages. MIT Press, Cambridge (1993)
6. Kleene, S. C.: Introduction to Metamathematics. Van Nostrand, New York (1952)
7. Nikitchenko M., Kryvolop A.: Semantic properties of monotone Floyd-Hoare logics. Bulletin of Taras Shevchenko National University of Kyiv Series: Physics & Mathematics, 3, pp. 215– 222 (in Ukrainian) (2012)
8. Nikitchenko M., Shkilnyak S.: Mathematical logic and theory of algorithms. Publishing house of Taras Shevchenko National University of Kyiv, Kyiv, 528 p. (In Ukrainian) (2008)
9. Blamey, S., Partial Logic, In: Gabbay D., Guenthner F. (Eds.), Handbook of Philosophical Logic, Volume III, D. Reidel Publishing Company, Dordrecht (1986)
10. Spivey M.: The Z Notation: A Reference Manual, 2nd edition. Prentice Hall International Series in Computer Science (1992)
11. Abrial J.-R.: The B-Book: Assigning Programs to Meanings. Cambridge University Press (1996)
12. Lamport L.: Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers. Addison-Wesley (2002)
13. George C., Haxthausen A.E., Hughes S., et al.: The RAISE Development Method. Prentice Hall, London (1995)
14. Gurevich Y.: Evolving Algebras 1993: Lipari Guide, In: E. Börger (Ed.), Specification and Validation Methods. pp. 9-36, Oxford University Press (1995)
15. Nikitchenko M., Tymofieiev V.: Satisfiability and Validity Problems in Many-sorted Composition-Nominative Pure Predicate Logics. In: V. Ermolayev et al. (eds.): ICTERI 2012, CCIS 347, pp. 89–110. Springer, Heidelberg (2012).
16. Nikitchenko M.S., Tymofieiev V.G.: Satisfiability in Composition-Nominative Logics. Central European Journal of Computer Science, vol. 2, issue 3, pp. 194-213 (2012)
17. Shkilniak S. S.: First-order logics of quasiary predicates. Kibernetika I Sistemnyi Analiz, 6, 32-50 (in Russian) (2010)