Cent. Eur. J. Comp. Sci. • 2(3) • 2012 • 194-213
DOI: 10.2478/s13537-012-0027-3

VERSITA

Central European Journal of **Computer Science**

Satisfiability in composition-nominative logics

Review Article

Mykola S. Nikitchenko*, Valentyn G. Tymofieiev†

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

Received 14 February 2012; accepted 17 August 2012

Abstract: Composition-nominative logics are algebra-based logics of partial predicates constructed in a semantic-syntactic style on the methodological basis, which is common with programming. They can be considered as generalizations of traditional logics on classes of partial predicates that do not have fixed arity. In this paper we present and investigate algorithms for solving the satisfiability problem in various classes of composition-nominative logics. We consider the satisfiability problem for logics of the propositional, renominative, and quantifier levels and prove the reduction of the problem to the satisfiability problem for classical logics. The method developed in the paper enables us to leverage existent state-of-the-art satisfiability checking procedures for solving the satisfiability problem in composition-nominative logics, which could be crucial for handling industrial instances coming from domains such as program analysis and verification. The reduction proposed in the paper requires extension of logic language and logic models with an infinite number of unessential variables and with a predicate of equality to a constant.

Keywords: partial predicates • partial logics • first-order logics • satisfiability • validity • compositionality • nominativity • compostion-nominative logics

1. Introduction

The satisfiability problem is one of the classical problems in logic [17]. Lately, interest in this problem has increased due to the practical value it has obtained in such areas as program verification, synthesis, analysis, testing, etc. [13, 15, 19]. In this paper we address the satisfiability problem in the context of the *composition-nominative approach* [20], which aims to construct a hierarchy of logics of various abstraction and generality levels on the methodological basis, which is common with programming. The main principles of the approach are principles of *development from abstract to concrete*, *priority of semantics*, *compositionality*, and *nominativity*.

These principles specify a hierarchy of new logics that are semantically based on algebras of predicates. Predicates are considered as partial mappings from a certain class of data *Dt* into the class of Boolean values *Bool*. Operations over

* *E-mail: nikitchenko@unicyb.kiev.ua (Corresponding author)*
† *E-mail: tvalentyn@univ.kiev.ua*

predicates are called *compositions*. They are treated as predicate construction tools. Data classes are considered on various abstraction levels but the main attention is paid to the class of *nominative data*. Such data consist of *name-value* pairs. In the simplest case nominative data can be considered as partial mappings from a certain set of names (variables) V into a set of basic (atomic) values A. These data are called *nominative sets*; their class is denoted ${}^{V}A$. Nominative sets represent program states for simple programming languages (see, for example, [18, 20, 27, 30]). Partial predicates and functions over ${}^{V}A$ are called *quasiary*, their classes are denoted $Pr^{V,A} = {}^{V}A \xrightarrow{p} Bool$ and $Fn^{V,A} = {}^{V}A \xrightarrow{p} A$ respectively. Partial mappings of type ${}^{V}A \xrightarrow{p} {}^{V}A$ are called *bi-quasiary*. Such mappings represent program semantics for simple programming languages; therefore their class is denoted $Prg^{V,A}$. From this follows that semantic models of programs and logics are mathematically based on the notion of nominative set (nominative data in general case). This fact permits the integration of models of programs and logics representing them as a hierarchy of composition-nominative models [22, 23]. Logics developed within such approach are called *composition-nominative logics* (CNL) because their predicates and functions are defined on classes of nominative data, and logical connectives and quantifiers are formalized as predicate compositions.

CNL can be considered as a generalization of classical predicate logic but for all that many methods developed within classical logic can also be applied to CNL. In the paper we confirm this statement for the satisfiability problem in CNL. We consider three levels of CNL – propositional, renominative, and quantifier levels – and construct the algorithms that reduce the satisfiability problem to classical cases, respectively, to the same problem in classical propositional logic, quantifier-free predicate logic, and classical first-order predicate logic. In the latter case the logic language should be extended with an infinite number of unessential variables and with additional predicate of equality to a constant.

The paper is structured as follows: In Section 2 we give an introduction to the composition-nominative approach, its main motivation, ideas, and notions taking into consideration that the literature on CNL is available primarily in Russian/Ukrainian. In Section 3 we give an overview of the composition-nominative logics hierarchy; then in Section 4 we give formal definitions of logics considered in this paper, and define the satisfiability problem in Section 5. In Sections 6–8 we discuss and prove reduction methods for solving the satisfiability problem on propositional, renominative, and quantifier levels respectively. We discuss related work in Section 9. Finally, in Section 10 we summarize our results and formulate directions for future investigations.

2. Composition-nominative approach to software system formalization

Mathematical logic proposes a powerful instrument for studying properties of software systems. Still, the application of existing or modified logics to software system domain is not easy. An analysis can demonstrate certain discrepancies between a problem to be solved and a logic being used. For example, in the development of software systems (later simply referred to as programs) we have to admit the following discrepancies:

- semantics of programs is adequately represented by partial functions whereas in traditional logic total functions and predicates are usually considered;
- programming languages have a developed system of data types whereas traditional logic prefers to operate with simple unstructured types (sorts);
- semantic aspects of programs prevail over syntactic aspects whereas in traditional logic we have an inverse situation.

These types of discrepancies complicate the usage of logic for program development, analysis, and verification. Therefore we advocate another scheme of relationship between mathematical logic and programming. Namely, we propose to take program models as an initial point and to construct logics based directly on such models. Thus, instead of adaptation of logic to program models we will “extract” logics from such models.

To realize this idea we should first construct adequate models of programs. To tackle this problem we use composition-nominative approach to program formalization [20], which aims to construct a hierarchy of program models of various abstraction and generality levels. The main principles of the approach are the following.

- *Development principle* (from abstract to concrete): program notions should be introduced as a process of their development that starts from abstract understanding, capturing essential program properties, and proceeds to more concrete considerations.

- *Principle of priority of semantics over syntax*: program semantic and syntactic aspects should be first studied separately, then in their integrity in which semantic aspects prevail over syntactic ones.
- *Compositionality principle*: programs can be constructed from simpler programs (functions) with the help of special operations, called compositions, which form a kernel of program semantics structures.
- *Nominativity principle*: nominative (naming) relations are basic ones in constructing data and programs.

Here we have formulated only principles relevant to the topic of the article; a richer system of principles is developed in [22]. The above-stated principles specify program models as *composition-nominative systems* (CNS) [20, 22]. 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, composition system can be specified as 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 (descriptive system), and ordinary procedure of term interpretation gives a denotation system.

CNS can be used to construct formal models of various programming, specification, and database languages [3, 20, 22]. Program models represented by CNS are mathematically simple, but specify program semantics rather adequately; program models are highly parametric and can represent programs of various abstraction and generality levels in a natural way; on the base of CNS there is a possibility to introduce the notion of special (abstract) computability and various axiomatic formalisms [21–23, 28].

CNS are classified in accordance with levels of abstraction of their parameters: data, functions, and compositions. Here we restrict ourselves to program model levels that are induced by abstraction levels of data.

Data are considered at three levels: abstract, Boolean, and nominative. At the abstract level data are treated as "black boxes", thus no information can be extracted. At the Boolean level new data considered as "white boxes" are added. Usually, these are logical values T (true) and F (false) that form the set *Bool*. At the nominative level data are considered as "grey boxes", constructed of "black" and "white boxes" with the help of naming relations. The last level is the most interesting for programming. Data of this level are called *nominative data*. The class of *nominative data* over a set of names V and a class of basic values W can be defined inductively, or equivalently, as the least fixed point of the recursive definition $ND(V, W) = W \bigcup (V \xrightarrow{m} ND(V, W))$, where $V \xrightarrow{m} ND(V, W)$ is the class of partial multi-valued (non-deterministic) functions.

For nominative data representation we use the form $d = [v_i \rightarrow a_i \mid i \in I]$. *Nominative membership relation* is denoted by $\in_n$. Thus, $v_i \rightarrow a_i \in_n d$ means that the value of v_i in d is defined and is equal to a_i; this can be written in another form as $d(v_i)\downarrow = a_i$. Let us note that a partial multi-valued function f cannot be precisely determined by its graph (the set of pairs (a, b) such that $f(a)\downarrow = b$). The reason is that for an argument a the function f can yield different values ($f(a) \downarrow = b_1$, $f(a) \downarrow = b_2$, etc.), and be undefined ($f(a) \uparrow$) at the same time.

The class $ND(V,W) \setminus W$ is called the class of *proper nominative data*, or *hierarchical nominative data*; data from the class $V \xrightarrow{m} W$ will be called *flat nominative data*, or *nominative sets*.

Concretizations of nominative data can represent various data structures, such as records, arrays, lists, relations, etc. [3, 20]. For example, a set $\{s_1, s_2, ..., s_n\}$ can be represented as a nominative set (flat nominative data or partial multi-valued function) $[1 \rightarrow s_1, 1 \rightarrow s_2, ..., 1 \rightarrow s_n]$, where 1 is treated as a standard name. Thus, we can formulate the following *data representation principle*: program data can be represented as concretizations of nominative data [22].

The three levels of data considered above specify three levels of semantics-based program models: abstract, Boolean, and nominative. Program models of the abstract level are very poor (actually, only sequencing compositions can be defined). Program models of the Boolean level are richer and permit to define structured programming constructs (sequence, conditional selection, and iteration). This level is still too abstract and does not explicitly specify data variables. At last, models of the nominative level permit to formalize compositions of traditional programming. This level involves variables of different types. Consider, for example, a simple educational programming language WHILE [18], which is based on three main syntactic components: arithmetic expressions, Boolean expressions, and statements. States of WHILE programs are considered as partial single-valued functions from the set of variables V to the set of values A. The class of all states $V \xrightarrow{p} A$ is also denoted by ${}^V A$. Thus, the semantics of WHILE components is the following: arithmetic expressions specify partial functions of the type ${}^V A \xrightarrow{p} A$ (quasiary functions), Boolean expressions define partial functions of the type ${}^V A \xrightarrow{p} Bool$ (quasiary predicates), statements specify functions of the type ${}^V A \xrightarrow{p} {}^V A$

(bi-quasiary functions). Note that in our terminology ${}^{V}A$ is a class of single-valued flat nominative data (nominative sets).

Example 1.

Consider a Boolean expression $x<y$. Its semantics can be formalized as a partial quasiary predicate *less*: ${}^{V}Z \xrightarrow{p} Bool$. This predicate is undefined on a flat nominative data $[x\to5,\ u\to4]$ (we write $less([x\to5,\ u\to4])\uparrow$), is defined on $[x\to5,\ u\to4,\ y\to2]$ with value F (we write $less([x\to5,\ u\to4,\ y\to2])\downarrow= F$). Note that if a value of *less* is defined on some data, then the predicate is defined with the same value on any extension of these data. This property is called *equitonicity* (a special case of monotonicity). Thus, $less([x\to5,\ u\to4,\ y\to2,\ v\to4])\downarrow=F$, $x, u, y, v \in V$. A specific new composition, which can be defined on this level, is renomination $R^{v_1,\dots,v_n}_{x_1,\dots,x_n}$ (formal definition is given in Subsection 4.2). For example, when we ask how to represent a formal model of $y<v$ given a formal model *less* of $x<y$, the answer will be $R^{x,y}_{y,v}(less)$. The constructed predicate can be evaluated in the expected manner, e.g.

$$R^{x,y}_{y,v}(less)([x \to 5, u \to 4, y \to 2, v \to 4]) \ = \ less([x \to 2, u \to 4, y \to 4, v \to 4]) \ = \ T.$$

Note that renomination (primarily in syntactic aspects) is widely used in classical logic, lambda-calculus, and specification languages like Z-notation [29], B [1], TLA [14], etc.

The notion of quasiary predicate can also be easily understood when we analyze Tarski's definition of first-order language semantics [17]. This semantics is based on the notion of interpretation which consists of two parts: 1) interpretation of predicate and function symbols in some structure, and 2) interpretation of individual variables in the domain of this structure. The latter are usually called variable assignments (or valuations) and can be represented by total mappings from a set of individual variables (names) V into some set of basic values A. The class of such total mappings will be denoted $V \xrightarrow{t} A$ or A^V , and called total nominative sets. Thus, Tarski's semantics interprets predicate and function symbols as total quasiary predicates and functions defined on A^V. In applications like model checking, program verification, automated theorem proving, etc., partial assignments (nominative sets) are often used instead of total assignments. This means that predicate and function symbols can be interpreted as partial functions defined on the class ${}^{V}A$ of nominative sets with values in *Bool* and A respectively, i.e. as quasiary predicates and functions.

More elaborate programming languages work with hierarchical nominative data. Composite names like $x_1.x_2...x_n$ are used to access data components in such languages. Such data can represent complex data structures used in programming.

3. Hierarchy of composition-nominative logics

Having described program models of various abstraction levels we can now start developing semantics-based logics which correspond to such models. Obtained logics will be called *composition-nominative logics* (CNL). Analysis of constructed program models shows that the main semantic notion of mathematical logic - the notion of predicate - can be defined at the Boolean level. At this level predicates are considered as partial functions from a class of abstract data *Dt* to *Bool*. In this case such compositions as disjunction $\vee$, negation $\neg$, etc., can be defined. These compositions are derived from Kleene's strong connectives [12] when partiality of predicates is taken into consideration. Thus, the main semantic objects are classes of algebras of partial predicates of the form $<Dt \xrightarrow{p} Bool; \vee, \neg >$. The obtained logics may be called *propositional logics of partial predicates*. Such logics are rather abstract, therefore their further development is required at the nominative level. As was mentioned earlier, at this level we have two sublevels determined respectively by flat and hierarchical nominative data.

Three kinds of logics can be constructed from program models on the flat nominative data level:

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 $Prg^{V,A}$.

For logics of pure quasiary predicates (*pure* CNL) we identify renominative, quantifier, and quantifier-equational levels. *Renominative* logics [23] are the most abstract among above-mentioned logics. The main new compositions for these logics are the compositions of renomination (renaming) of the form $R^{v_1,\dots,v_n}_{x_1,\dots,x_n}$: $Pr^{V,A} \xrightarrow{t} Pr^{V,A}$. Intuitively, given a quasiary

predicate p and a nominative set d the value of $R^{v_1,\dots,v_n}_{x_1,\dots,x_n}(p)(d)$ is evaluated in the following way: first, a new nominative set d' is constructed from d by changing the values of the names $v_1,\dots,v_n$ in d to the values of the names $x_1,\dots, x_n$ respectively; then the predicate p is applied to d'. The obtained value (if it was evaluated) will be the result of $R^{v_1,\dots,v_n}_{x_1,\dots,x_n}(p)(d)$. For simplicity's sake we will also use simplified notation $R^{\bar{v}}_{\bar{x}}$ for renomination composition. The basic composition operations of renominative logics are $\vee$, $\neg$ and $R^{\bar{v}}_{\bar{x}}$.

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 $S^{\bar{x}}$ (see [20, 23]), which formalize substitution of functions into predicate (or function). It seems also natural to introduce special 0-ary compositions, called denaming functions $'x$. They can be considered a parametric unary function with one parameter x. Given a nominative set, $'x$ yields a value of the name x in this set. 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 $\vee$, $\neg$, $S^{\bar{x}}$, $\exists x$, and $'x$.

At the function-equational level, a special equality composition $=$ can be introduced additionally [23]. The basic compositions of logics of the function-equational level are $\vee$, $\neg$, $S^{\bar{x}}$, $\exists x$, $'x$, 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, equicompatible, etc. [23]. 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 called *maxitotal equitone*; if a predicate p is a restriction of some equitone predicate then p is *equicompatible* predicate. Logics based on maxitotal equitone, equitone, and equicompatible predicates are the "closest" generalization of the classical first-order logic that preserves its main properties. These logics are called *neoclassical logics* [23].

The level of *program logics* is quite rich. First, program compositions that describe the structure of programs are defined. In the simplest case these are:

1. parametric assignment composition $AS^x{:}Fn^{V,A} \xrightarrow{t} Prg^{V,A}$,
2. composition of sequential execution $\bullet{:}Prg^{V,A} \times Prg^{V,A} \xrightarrow{t} Prg^{V,A}$,
3. conditional composition IF: $Pr^{V,A} \times Prg^{V,A} \times Prg^{V,A} \xrightarrow{t} Prg^{V,A}$,
4. cycling composition WH: $Pr^{V,A} \times Prg^{V,A} \xrightarrow{t} Prg^{V,A}$.

Then we should define compositions specifying program properties. Here we only mention a composition that formalizes the notion of assertion in *Floyd-Hoare logic*. From the semantic point of view an assertion scheme of the form $\{p\}prog\{q\}$ may be considered as composition *FH,* which given two quasiary predicates p (precondition), q (postcondition), and a bi-quasiary function (a program) *prog* produces new quasiary predicate denoted by $FH(p, prog, q)$. Such constructions nicely correspond with main approaches to formal semantics of programs, in particular, with denotational semantics, which treats program operators as state function compositions [18, 27, 30].

An example of simple three-sorted predicate-function-program algebra is presented in Fig. 1.

Classes of terms of this algebra may be considered as sets of formulas (or their components) of corresponding logics. The Fig. 1 demonstrates that composition-nominative approach aims to construct logics based on various classes of program models. For a number of such logics axiomatic calculi were constructed and their properties were investigated [23, 28].

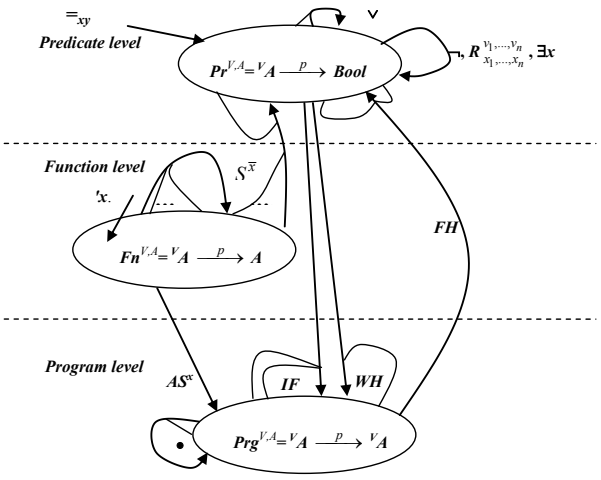

Figure 1. Simple three-sorted predicate-function-program algebra.

4. Composition-nominative logics: formal definitions

Here we consider only pure CNL of the following abstraction levels: *propositional*, *renominative*, and *quantifier*. Each level specifies classes of predicates, defined on classes of data, and a set of compositions that correspond to this level. This means that a class of predicate algebras should be specified for each level. Such algebras form the semantic base for CNL of chosen level. Predicate symbols are interpreted as predicates, and formulas are considered as terms, interpreted in such algebras.

Let $Bool=\{F,T\}$ be a set of Boolean (logical) values, $Pr(Dt)=Dt\xrightarrow{p}Bool$ be the set of all partial predicates specified on some data set Dt, C be a set of total n-ary compositions of predicates of the type $Pr(Dt)^n \xrightarrow{t} Pr(Dt)$. Then a pair $<Pr(Dt), C>$ is *an algebra of partial predicates*. In terms of traditional logic the set Dt is interpreted as a possible world or universe, elements of Dt are treated as world states, predicates from $Pr(Dt)$ are considered as world state properties, and compositions from C are treated as operators over $Pr(Dt)$.

Parameters of logics (classes of predicates, interpretations of predicate symbols, and languages) can be restricted.

Examples of restricted predicate classes are total predicates, equitone predicates, etc. Interpretations can also be restricted, for example some predicate symbols in the language can have a fixed interpretation. A restricted language can consist only of formulas that are in some special normal form. Thus, we will obtain logics of total predicates, logics with fixed interpretations, logics with restricted classes of formulas. Such logics are called restricted.

These considerations lead to the following definitions of the notion of concrete logic.

A concrete composition-nominative logic Lg is specified by

1. a signature (Cs, Ps), where Cs is a set of composition symbols and Ps is a set of predicate symbols;
2. a certain class of data sets; an element of this class is usually denoted by Dt;
3. a certain class of algebras of the form $<Prr, C>$, where $Prr\subseteq Pr(Dt)$, Dt is one of data sets from the class specified for Lg, and operations (compositions) C are interpretations of Cs in this algebra;
4. a certain class of interpretations of predicate symbols of the form $I^{Ps}:Ps\xrightarrow{t} Prr$ for each algebra $<Prr, C>$;
5. a language of Lg, which is a subset of all formulas of the signature (Cs, Ps), constructed over predicate symbols from the set Ps with the help of symbols of compositions from the set Cs. The class of all formulas $Fr_{Lg}(Cs, Ps)$ can be defined inductively:
    1. if $P \in Ps$ then $P \in Fr_{Lg}(Cs, Ps)$;
    2. if $\Phi_1,\dots, \Phi_n \in Fr_{Lg}(Cs, Ps)$, $c \in Cs$ is an n-ary composition symbol then $c(\Phi_1,\dots, \Phi_n) \in Fr_{Lg}(Cs, Ps)$.

Note that for simplicity's sake we will use the same notation for compositions as algebraic operations (the set C) and their symbols in the language signature (the set Cs). The class $Fr_{Lg}(Cs, Ps)$ is exactly the class of all terms of algebras specified for Lg. In the sequel we consider formulas of Fr_{Lg} (Cs, Ps) in their traditional form using infix operations and brackets; brackets can be omitted according to common rules for the priorities of operations (priority of binary disjunction is weaker than priority of unary operations).
From these definitions follows that the main semantic object in a CNL Lg is a class of algebras of the form $<Prr, C>$, where $Prr \subseteq Pr(Dt)$ for some Dt. In CNL compositions have a fixed interpretation. Usually, the set of compositions is clear from the context, therefore we will represent such algebras by a pair (Dt, Prr), which is called *α-interpretation*. Interpretations of predicate symbols of the form $I^{Ps}{:}Ps \xrightarrow{t} Prr$ in an algebra $<Prr, C>$ will be called *σ-interpretations*. If Ps is clear from the context we write I. A combination of an α-interpretation and a σ-interpretation gives a *π-interpretation* denoted (Dt, Prr, I^{Ps}). Such interpretations may also be called models of a logic. Given a π-interpretation $J=(Dt, Prr, I^{Ps})$ any formula $\Phi \in Fr_{Lg}$ (Cs, Ps) by usual procedure of term interpretation can be interpreted as a certain predicate denoted $\Phi_J \in Prr$. Prefixes π-, α-, σ- may be omitted.
Now we will give formal definitions for three types of composition-nominative logics considered on propositional, renominative, and quantifier levels. Logics of such types will be denoted as PCNL, RCNL, and QCNL respectively. We will consider general (unrestricted) pure CNL (without function symbols).
In definitions we will use the following notation:

1. $p(d)\downarrow= b$ means that a predicate p is defined on data d with a Boolean value b;
2. $p(d)\uparrow$ means that a predicate p on d is undefined.

4.1. Composition-nominative logic of propositional level

Logics of propositional level are called *propositional composition-nominative logics* (PCNL). An unrestricted concrete logic L_P of this level is determined by two basic compositions: disjunction $\vee$ and negation $\neg$. Thus, the signature of L_P is $(\{\vee, \neg\}, Ps)$. Any data set Dt belongs to the class of data sets specified for L_P, thus semantic base of L_P is the class of algebras of the form $AP(Dt)= <Pr(Dt), \vee, \neg >$.
Basic compositions are Kleene's disjunction $\vee$ and negation $\neg$, defined by the following formulas ($p, q \in Pr(Dt)$, $d \in Dt$):

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

$$(\neg p)(d) = \begin{cases} T, \text{ if } p(d)\downarrow= F, \\ F, \text{ if } p(d)\downarrow= T, \\ \text{undefined if } p(d)\uparrow. \end{cases}$$

Language of L_P, represented by the class of formulas $Fr_{PCNL}(\{\vee, \neg\}, Ps)$, is defined inductively:

1. If $P \in Ps$ then $P \in Fr_{PCNL}(\{\vee, \neg\}, Ps)$. Such formula is called an atomic formula.
2. If $\Phi, \Psi \in Fr_{PCNL}(\{\vee, \neg\}, Ps)$ then $(\Phi \vee \Psi) \in Fr_{PCNL}(\{\vee, \neg\}, Ps)$ and $\neg\Phi \in Fr_{PCNL}(\{\vee, \neg\}, Ps)$.

4.2. Composition-nominative logic of renominative level

Logics of renominative level are called *renominative composition-nominative logics* (RCNL). An unrestricted concrete logic L_R of this level is determined by three basic compositions: disjunction $\vee$, negation $\neg$, and renomination $R^{\bar{v}}_{\bar{x}}$: $Pr({}^V A) \xrightarrow{t} Pr({}^V A)$, where $\bar{v} = (v_1, ..., v_n)$ and $\bar{x} = (x_1, ..., x_n)$ are lists of variables from a certain set V, fixed for L_R; variables from $\bar{v}$ are called *upper names of renomination composition* and should be distinct, variables from $\bar{x}$ are called *lower names of renomination composition*, $n \geq 0$. Thus, the signature of L_R is $(V, \{\vee, \neg, R^{\bar{v}}_{\bar{x}}\}, Ps)$. Please note that $R^{\bar{v}}_{\bar{x}}$ is a parametric composition, which represents a class of renomination compositions with different parameters constructed from elements of V. It means that the set of composition symbols actually is $\{\vee, \neg\} \cup \{R^{\bar{v}}_{\bar{x}} | \bar{x}, \bar{v} \in V^*$ and

$\bar{x}, \bar{v}$ satisfy the restrictions on parameters of renomination composition}. This notational convention concerns signatures of other logics with parametric compositions considered in the paper; therefore we include the set of names into such signatures. For every set A, the set ${}^{V}A$ of all nominative sets belongs to the class of data sets specified for L_R. Semantic base of L_R are all algebras of the form $AR\ (V, A)=<Pr\ ({}^{V}A), \vee, \neg\ , R^{\bar{v}}_{\bar{x}} >$. The set $Pr({}^{V}A)$ will also be denoted $Pr^{V,A}$. Unary parametric composition of *renomination* $R^{v_1,\dots,v_n}_{x_1,\dots,x_n}$: $Pr^{V,A} \xrightarrow{t} Pr^{V,A}$ is defined by the following formula ($p \in Pr^{V,A}$, $d \in {}^{V}A$):

$$(R^{v_1,\dots,v_n}_{x_1,\dots,x_n}\ p)\ (d) = p\left([v \mapsto a \in_n d | v \notin \{v_1, \dots, v_n\}] \nabla [v_i \mapsto d(x_i) |\ d(x_i)\downarrow, i \in \{1, \dots, n\}]\right).$$

The ∇ operation is defined as follows. If d_1 and d_2 are two nominative sets, then $d = d_1 \nabla d_2$ consists of all named pairs of d_2 and only those pairs of d_1, whose names are not defined in d_2. Note that $R\ p$ (when $n = 0$) is equal to p.
Language of L_R is represented by the class of formulas $Fr_{RCNL}(V, \{\vee, \neg, R^{\bar{v}}_{\bar{x}}\}, Ps)$. As V defines the set of composition symbols Cs for $RCNL$, we also use a simplified notation $Fr_{RCNL}(V, Ps)$. This class is defined inductively by rules 1 and 2 from the definition of $Fr_{PCNL}(\{\vee, \neg\}, Ps)$ in which $Fr_{PCNL}(\{\vee, \neg\}, Ps)$ is changed on $Fr_{RCNL}(V, Ps)$ and by the following new rule:

3. If $\bar{v} = (v_1, \dots, v_n)$, $\bar{x} = (x_1, \dots, x_n)$, $\bar{v}$ is a list of distinct variables, $v_i, x_i \in V$ for all $i \in \{1, \dots, n\} (n \geq 0)$, $\Phi \in Fr_{RCNL}(V, Ps)$ then $R^{\bar{v}}_{\bar{x}}\Phi \in Fr_{RCNL}(V, Ps)$.

4.3. Composition-nominative logic of quantifier level

Logics of quantifier level are called *quantifier composition-nominative logics* (QCNL). An unrestricted concrete logic L_Q of this level is determined by four basic compositions: disjunction $\vee$, negation $\neg$, renomination $R^{\bar{v}}_{\bar{x}}$, and existential quantification $\exists x$, where $x \in V$, $\bar{x}, \bar{v}$ are vectors of variables from V. Thus, the signature of L_Q is $(V, \{\vee, \neg, R^{\bar{v}}_{\bar{x}}, \exists x\}, Ps)$. Let us note that elements from the set of names V are used as parameters of existential quantification (and renomination too).
The class of data sets specified for L_Q is the same as one specified for L_R. Semantic base of L_Q are all algebras of the form $AQ(V, A)=<Pr^{V,A}, \vee, \neg, R^{\bar{v}}_{\bar{x}}, \exists x >$.
On the quantifier level predicates can be applied to all data obtained from given data by changing the basic values of the component with a fixed name. Therefore a composition of existential quantification $\exists x$ with the parameter $x \in V$ is defined by the following formula ($p \in Pr^{V,A}$, $d \in {}^{V}A$):

$$(\exists x\ p)(d) = \begin{cases} T, & \text{if } b \in A \text{ exists}: \ p(d\nabla x \mapsto b)\downarrow = T, \\ F, & \text{if } p(d\nabla x \mapsto a)\downarrow = F \text{ for each } a \in A, \\ & \text{undefined in other cases.} \end{cases}$$

Here $d\nabla x \mapsto a$ is a shorter form for $d\nabla[x \mapsto a]$.
Language of L_Q, represented by the class of formulas $Fr_{QCNL}(V, Ps)$, is defined inductively by rules 1-3 from the definition of $Fr_{RCNL}(V, Ps)$ in which $Fr_{RCNL}(V, Ps)$ is changed to $Fr_{QCNL}(V, Ps)$ and by the following new rule:

4. If $x \in V$, $\Phi \in Fr_{QCNL}(V, Ps)$ then $\exists x\Phi \in Fr_{QCNL}(V, Ps)$.

Let us note that for all described logics derived compositions (such as conjunction $\wedge$, universal quantification $\forall x$, etc.) are defined in a traditional way.
In this paper we develop reductive methods for solving satisfiability problem. It means that we reduce this problem to the satisfiability problem in classical logics. So, we will also be referring to several fragments of classical first-order logic. We consider the pure first-order logic, that is, classical first-order logic without function symbols.
Now we give definitions of three classical logics: propositional logic, quantifier-free predicate logic, and first-order predicate logic. We will give their definitions in the style of CNL describing only distinctions from corresponding CNL.

4.4. Classical propositional logic

Logics of propositional level are called *classical propositional logics* (PCL). A concrete logic L_{PCL} of this level is specified by the signature $(\{\vee, \neg\}, Ps)$. A class of data sets consists only of one empty set $\emptyset$. In this case we should specify a class of predicates $\emptyset \xrightarrow{t} Bool$. For simplicity's sake we identify $\emptyset \xrightarrow{t} Bool$ with $Bool$. This is a disputable solution but we follow a mathematical tradition of identifying null-ary predicates with Boolean constants. Of course we should ensure that all definitions agree with this solution. Accordingly, definitions of compositions should also take this fact into account. This solution gives us only one algebra $APCL = <Bool, \vee, \neg >$ and only one class of σ-interpretations of the form I^{Ps}_{PCL}: $Ps \xrightarrow{t} Bool$. The language $Fr_{PCL}(\{\vee, \neg\}, Ps)$ is the same as $Fr_{PCNL}(\{\vee, \neg\}, Ps)$.

4.5. Classical quantifier-free predicate logic

Logics of quantifier-free level are called here *classical quantifier-free logics* (FCL). A concrete logic L_{FCL} of this level is specified by the signature $(V, \{\vee, \neg\}, Ps, arity)$ where V is a set of variables, and $arity$: $Ps \xrightarrow{t} \{0,1,2, \dots\}$ is a function that for each predicate symbol yields its arity.
For every set A the set $A^V = V \xrightarrow{t} A$ of all total nominative sets belongs to the class of data sets specified for L_{FCL}. A class of total predicates $TPr(A^V) = A^V \xrightarrow{t} Bool$ is used for interpretation of formulas. The semantic base of FCL are algebras of the form $AFCL(V,A) = <TPr(A^V), \vee, \neg >$. Interpretations of predicate symbols are also defined differently from those in RCNL. They are mappings

$$I^{Ps}_{CL} : Ps \xrightarrow{t} \bigcup_{n\geq 0} \left(A^n \xrightarrow{t} Bool \right)$$

such that $I^{Ps}_{CL}(P) \in A^n \xrightarrow{t} Bool$ if $arity(P) = n$ for $P \in Ps$. Thus, interpretations have the form $K^{Ps}_{CL} = (A^V, TPr(A^V), I^{Ps}_{CL})$. Such interpetations will be called κ-interpetations.
The language $Fr_{FCL}(V, Ps)$ is defined inductively:

1. If $P \in Ps$, $arity(P) = n$, and $x_1, \dots, x_n \in V$, then $P(x_1, \dots, x_n) \in Fr_{FCL}(V, Ps)$. Such formula is called an atomic formula.
2. If $\Phi, \Psi \in Fr_{FCL}(V, Ps)$ then $(\Phi \vee \Psi) \in Fr_{FCL}(V, Ps)$ and $\neg\Phi \in Fr_{FCL}(V, Ps)$.

A κ-interpretation $K^{Ps}_{CL} = (A^V, TPr(A^V), I^{Ps}_{CL})$, in a simplified form denoted by K, for every atomic formula $P(x_1, \dots, x_n)$ defines its meaning in $TPr(A^V)$ as a predicate $P(x_1, \dots, x_n)_K$ such that $P(x_1, \dots, x_n)_K(d) = I^{Ps}_{CL}(P)(d(x_1), \dots, d(x_n))$ for every $d \in A^V$. The meaning Φ_K of any formula $\Phi \in Fr_{FCL}(V, Ps)$ is defined in a usual way.

4.6. Classical first-order predicate logic

Logics of quantifier level are called *classical first-order predicate logics* (QCL). A concrete logic L_{QCL} of this level is specified in the same way as L_{FCL} with such distinctions: the signature is $(V, \{\vee, \neg, \exists x\}, Ps, arity)$; considered algebras are $AQCL = <TPr(A^V), \vee, \neg, \exists x >$; for defining $Fr_{QCL}(V, Ps)$ we use an additional rule

3. If $x \in V$, $\Phi \in Fr_{QCL}(V, Ps)$ then $\exists x\Phi \in Fr_{QCL}(V, Ps)$.

The logics PCL, FCL, and QCL correspond to PCNL, RCNL, and QCNL respectively.

5. Satisfiability problem for CNL

Let $\Phi \in Fr_{Lg}(Cs, Ps)$. We assume that in the algebras of the form $<Prr, C>$ compositions from C are interpretations of composition symbols Cs. We also assume that a set of predicate symbols Ps is fixed throughout this section. Formula Φ is called *satisfiable in a π-interpretation* $J = (Dt, Prr, I)$ if there is a $d \in Dt$ such that $\Phi_J(d)\downarrow = T$. We shall denote this by $J \mid\approx \Phi$. Let us note that in classical predicate logic d is called *variable valuation* or *variable assignment*. A CNL formula Φ is called *satisfiable in a predicate class Prr* (we write $(Dt, Prr) \mid\approx \Phi$) if there exists an interpretation

$J = (Dt, Prr, I)$ in which Φ is satisfiable. A CNL formula Φ is called *satisfiable* if there exists an interpretation J in which Φ is satisfiable. We shall denote this as $|\approx \Phi$. We call formulas Φ and Ψ *equisatisfiable* if they are either both satisfiable or both not satisfiable (i.e. unsatisfiable).
Satisfiability of a formula is related to its validity. A CNL formula Φ is called *valid in a π-interpretation* $J = (Dt, Prr, I)$ if there is no $d \in Dt$ such that $\Phi_J(d)\downarrow = F$. We shall denote this as $J \models \Phi$, which means that Φ is not refutable in J. Such definition of validity is chosen in accordance with the principle of development from abstract to concrete. This is the simplest (the most abstract) treatment of validity, which specifies only two cases: "valid" and "not valid". A CNL formula Φ is called *valid in a predicate class Prr* (we write $(Dt, Prr) \models \Phi$) if for every interpretation $J = (Dt, Prr, I)$ we have $J \models \Phi$. Formula Φ is called *valid* if $J \models \Phi$ for every interpretation J. We call formulas Φ and Ψ *equivalent* if $\Phi_J = \Psi_J$ for every interpretation J.
If we need to emphasize that a formula Φ belongs to the language of specific logic (e.g. PCNL, QCNL, FCL, etc.), we will use the corresponding subscript: (e.g. Φ_{PCNL}, Φ_{QCNL}, Φ_{FCL} etc.). When needed we will include the corresponding logic abbreviation in the satisfiability sign $|\approx$, e.g. $|\approx_{PCNL}$, $|\approx_{QCNL}$, $|\approx_{FCL}$, etc.
Validity and satisfiability are related with each other. But due to possible presence of a nowhere defined predicate (which is a valid predicate) we do not have in CNL the property that Φ is satisfiable if Φ is valid (which holds for classical first-order logic). But reduction of satisfiability to validity still holds in CNL. We formulate this statement for every concrete CNL Lg (see definitions in Section 4).

Lemma 5.1.

Let $\Phi \in Fr_{Lg}(Cs, Ps)$ and $<Pr(Dt), C>$ be an algebra of partial predicates, $J=(Dt, Pr(Dt), I)$ be a π-interpretation. Formula Φ is satisfiable in J iff $\neg\Phi$ is not valid in J.

Consider an algebra $AP(Dt)=<Pr(Dt), C>$. We say that $p \subseteq q$, where $p, q \in Pr(Dt)$, if for every $d \in Dt$ such that $p(d)\downarrow = b$ we have $q(d)\downarrow = b$ (this means inclusion of predicate graphs). We say that $(p_1,..., p_n) \subseteq (q_1,..., q_n)$ if $p_i \subseteq q_i$ for every $i \in \{1, ..., n\}$. An n-ary composition $c \in C$ is called *monotone* if from $(p_1,..., p_n) \subseteq (q_1,..., q_n)$ follows $c(p_1,..., p_n) \subseteq c(q_1,..., q_n)$.

Lemma 5.2.

Compositions in algebras of the forms $AP(Dt) = <Pr(Dt), \vee, \neg>$, $AR(V, A) = <Pr^{V,A}, \vee, \neg, R^{\bar{v}}_{\bar{x}}>$, and $AQ(V, A) = <Pr^{V,A}, \vee, \neg, R^{\bar{v}}_{\bar{x}}, \exists x>$ are monotone.

Proof.

The proof is straightforward. For example, let us show monotonicity of $\exists x$. Assume $p, q \in Pr^{V,A}$, $p \subseteq q$. Let $\exists x\, p(d)\downarrow = T$. Then there exists $b \in A$ such that $p(d\nabla x \mapsto b)\downarrow = T$; thus we have that $q(d\nabla x \mapsto b)\downarrow = T$; therefore $\exists x\, q(d)\downarrow = T$. Let $\exists x\, p(d)\downarrow = F$. Then $p(d\nabla x \mapsto a)\downarrow = F$ for every $a \in A$; thus we have that $q(d\nabla x \mapsto a)\downarrow = F$ for every $a \in A$. This means that $\exists x\, q(d)\downarrow = F$. □

Monotonicity of CNL compositions guarantees stability of satisfiability under predicate extensions.
Let $J_1 = (Dt, Pr(Dt), I_1)$, $J_2 = (Dt, Pr(Dt), I_2)$ be two interpretations such that $I_1(P) \subseteq I_2(P)$ for every $P \in Ps$. Then we say that J_2 extends J_1 and denote this by $J_1 \leq J_2$.

Lemma 5.3.

Let $\Phi \in Fr_{Lg}(Cs, Ps)$ and $<Prr, C>$ be an algebra of partial predicates. Let $J_1 = (Dt, Prr, I_1)$, $J_2 = (Dt, Prr, I_2)$ be two interpretations such that $J_1 \leq J_2$. Suppose that all compositions in C are monotone. Then from $J_1 |\approx \Phi$ follows $J_2 |\approx \Phi$.

Proof.

The statement can easily be proved by induction over the structure of formula Φ. □

From Lemma 5.3 follows that given a π-interpretation $J = (Dt, Prr, I)$ such that $J \mid\approx \Phi$ we can construct an interpretation $K = (Dt, TPr(Dt), I')$ within the class of total predicates such that $J \leq K$ and $K \mid\approx \Phi$.

Lemma 5.4.

Let $\Phi \in Fr_{Lg}(Cs, Ps)$. Then from $(Dt, Prr) \mid\approx \Phi$ follows $(Dt, TPr(Dt)) \mid\approx \Phi$.

For all three CNL levels considered in the paper the classes of total predicates are closed under compositions of respective levels.

Lemma 5.5.

The class $TPr(Dt) = Dt \xrightarrow{t} Bool$ of total predicates over Dt forms a sub-algebra in $AP(Dt)$;
the class $TPr({}^{V}A) = ({}^{V}A \xrightarrow{t} Bool)$ is a sub-algebra in $AR(V, A)$ and in $AQ(V, A)$.

6. Satisfiability for propositional CNL

Let $TCPr(Dt) \subseteq TPr(Dt)$ be a set of total constant predicates $TCPr(Dt) = \{Tp, Fp\}$ such that $Tp(d)\downarrow = T$ and $Fp(d)\downarrow = F$ for every $d \in Dt$.

Lemma 6.1.

Let $\Phi \in Fr_{PCNL}(\{\vee, \neg\}, Ps)$, $(Dt, TPr(Dt))$ be an α-interpretation. Then $(Dt, TPr(Dt)) \mid\approx \Phi$ iff $(Dt, TCPr(Dt)) \mid\approx \Phi$.

Proof.

To prove this lemma we need to show that for every π-interpretation $J = (Dt, TPr(Dt), I)$ such that $J \mid\approx \Phi$ we can construct a π-interpretation $K = (Dt, TCPr(Dt), I')$ such that $K \mid\approx \Phi$. From $J \mid\approx \Phi$ it follows that there exists $d \in Dt$ such that $\Phi_J(d)\downarrow = T$. For every $P \in Ps$ let $I'(P)$ be a total constant predicate taking the value $I(P)(d)$. It can easily be shown that $\Phi_K(d) \downarrow = T$, which means that $K \mid\approx \Phi$. The inverse direction holds because $TCPr(Dt) \subseteq TPr(Dt)$. □

Lemma 6.1 justifies considering only σ-interpretations in the set of total constant predicates for checking satisfiability of propositional CNL formulas. In fact, such reductions can be continued by considering an empty data set. In this case $TCPr(\emptyset) = Bool$ as was discussed earlier.

Lemma 6.2.

Let $\Phi \in Fr_{PCNL}(\{\vee, \neg\}, Ps)$, $d \in Dt$. Then $(Dt, TCPr(Dt)) \mid\approx \Phi$ iff $(\emptyset, TCPr(\emptyset)) \mid\approx \Phi$.
The proof is trivial. □

Summarizing formulated results we obtain the following chain of reductions:

$$(Dt, Pr(Dt)) \mid\approx \Phi \Leftrightarrow (Dt, TPr(Dt)) \mid\approx \Phi \Leftrightarrow (Dt, TCPr(Dt)) \mid\approx \Phi \Leftrightarrow (\emptyset, TCPr(\emptyset)) \mid\approx \Phi.$$

This essentially means that for checking satisfiability of Φ in CNL of the propositional level we can use the methods for checking satisfiability in the classical propositional logic. Thus, we have proved the following theorem.

Theorem 6.1.

Let $\Phi \in Fr_{PCNL}(\{\vee, \neg\}, Ps)$. Then $|\approx_{PCNL}\Phi$ iff $|\approx_{PCL}\Phi$.

For checking satisfiability in classical propositional logic many methods were developed [9]. Among them we would like to mention Davis-Putnam-Logemann-Loveland algorithm [8], variations of which are very widely used.

7. Satisfiability problem for renominative CNL

In this section we suggest a technique that allows reducing the satisfiability problem for CNL formulas of the renominative level to the satisfiability problem for quantifier-free formulas of classical predicate logic. First, we put an RCNL formula into a special equivalent representation in which renomination composition is applied only to predicate symbols, then we strengthen this representation to a unified renominative form, and at last, we apply syntactical transformation rules that construct an equisatisfiable formula of classical quantifier-free predicate logic.

A (sub)formula of the form $R^{\bar{v}}_{\bar{x}}P$ where $P \in Ps$ is called *a renominative atom*. An RCNL formula Φ is said to be in *renominative normal form* (RNF) if renomination compositions and predicate symbols occur in Φ only in renominative atoms. Recall that lists of names in renomination compositions may be empty (empty renomination). In this case renominative atoms have the form $R\ P$ $(P \in Ps)$. Here we assume that a non-empty renomination occurs in Φ; otherwise Φ can be trivially reduced to a propositional formula.

We consider a total mapping $rnf : Fr_{RCNL}(V, Ps) \xrightarrow{t} Fr_{RCNL}(V, Ps)$ that maps an RCNL formula to its RNF. Given an RCNL formula Φ, renominative normal form $rnf[\Phi]$ is constructed as described by rules R1-R7:

R1) $rnf[P] = R\ P$ $(P \in Ps)$

R2) $rnf[(\Phi_1 \vee \Phi_2)] = (rnf[\Phi_1] \vee rnf[\Phi_2])$

R3) $rnf[\neg\Phi] = \neg rnf[\Phi]$

R4) $rnf[R^{\bar{v}}_{\bar{x}}P] = R^{\bar{v}}_{\bar{x}}P$ $(P \in Ps)$

R5) $rnf[R^{\bar{v}}_{\bar{x}}(\Phi_1 \vee \Phi_2)] = (rnf[R^{\bar{v}}_{\bar{x}}\Phi_1] \vee rnf[R^{\bar{v}}_{\bar{x}}\Phi_2])$

R6) $rnf[R^{\bar{v}}_{\bar{x}}\neg\Phi] = \neg rnf[R^{\bar{v}}_{\bar{x}}\Phi]$

R7) $rnf[R^{v_1,\dots,v_n,w_1,\dots,w_m}_{x_1,\dots,x_n,y_1,\dots,y_m}\ R^{v_1,\dots v_n,u_1,\dots,u_k}_{s_1,\dots,s_n,z_1,\dots,z_k}\Phi] = rnf[R^{v_1,\dots,v_n,w_1,\dots,w_m,u_1,\dots,u_k}_{\alpha_1,\dots,\alpha_n,y_1,\dots,y_m,\beta_1,\dots,\beta_k}\Phi]$, where
$w_i \neq u_j$ $(i = 1,\dots,m; j = 1,\dots,k)$, $\alpha_i = s_i(v_1,\dots,v_n,w_1,\dots,w_m \,/\, x_1,\dots,x_n,\ y_1,\dots,y_m)$, $\beta_j = z_j(v_1,\dots,v_n,\ w_1,\dots,w_m \,/\, x_1,\dots,x_n,\ y_1,\dots,y_m)$.
Here $r(b_1,\dots,b_q \,/\, c_1,\dots,c_q) = r$ if $r \notin \{b_1,\dots,b_q\}$, $r(b_1,\dots,b_q \,/\, c_1,\dots,c_q) = c_i$ if $r = b_i$ for some i. This rule represents explicitly the result of function composition of parameters of two successive renominations.

The transformation $rnf : Fr_{RCNL}(V, Ps) \xrightarrow{t} Fr_{RCNL}(V, Ps)$ relies on equivalent formula transformations of RCNL [23], thus the following lemma holds.

Lemma 7.1.

Let $\Phi \in Fr_{RCNL}(V, Ps)$. Then $|\approx \Phi$ iff $|\approx rnf[\Phi]$.

Note that proposed way of construction of renominative normal form preserves the reciprocal order of propositional compositions in the original formula. Thus, the obtained formula is practically of the same complexity as the initial formula.

An RCNL formula Φ is said to be in *unified renominative normal form* (URNF) if it is in renominative normal form and for every pair of its renominative atoms $R^{\bar{u}}_{\bar{q}}P$ and $R^{\bar{w}}_{\bar{y}}Q$ we have that vectors $\bar{u}$ and $\bar{w}$ coincide; that is, in all renominative atoms the lists of their upper names are the same.

We will define a total multi-valued (non-deterministic) mapping $urnf : Fr_{RCNL} \xrightarrow{tm} Fr_{RCNL}$ that transforms an RCNL formula to its URNF. It means that given an RCNL formula Φ we non-deterministically construct $urnf[\Phi]$.
Let Φ be an arbitrary RCNL formula in renominative normal form. Denote by V_Φ the set of all names that occur as upper names in renominative atoms of Φ. To construct URNF we unify all renominative atoms as follows. First, we choose an ordering of variables (names) from V_Φ and get a list $\bar{v}$; then we transform all renominative atoms of the form $R^{\bar{u}}_{\bar{q}}P$ occurring in Φ to the form $R^{\bar{v}}_{\bar{x}}P$, so that $\bar{v}$ becomes the list of upper names in renomination compositions, while the lists of lower names may differ. This is done using the *identical renomination rule*: $R^{\bar{u}}_{\bar{q}}P = R^{z,\bar{u}}_{z,\bar{q}}P$, and *commutative renomination rule*: $R^{u_1,\ldots,\ u_i,\ldots,\ u_j,\ldots,\ u_n}_{q_1,\ldots,\ q_i,\ldots,\ q_j,\ldots,\ q_n}P = R^{u_1,\ldots,\ u_j,\ldots,\ u_i,\ldots,\ u_n}_{q_1\ldots,\ q_j,\ldots,\ q_i,\ldots,\ q_n}P$. For a formula Φ we will denote any possible URNF of Φ by $urnf[\Phi]$.

Lemma 7.2.

Let $\Phi \in Fr_{RCNL}(V, Ps)$. Then $|\approx \Phi$ iff $|\approx urnf[\Phi]$.

Proof.

Applications of identical and commutative renomination rules preserve equivalence of formulas, therefore the formulas Φ and $urnf[\Phi]$ are equivalent. Consequently, they are equisatisfiable. □

Now we describe a transformation $clf : Fr_{RCNL}(V, Ps) \xrightarrow{p} Fr_{FCL}(V, Ps)$ which, given an RCNL formula in URNF, yields a formula of quantifier-free classical logic as follows:

1. $clf[R^{v_1,\ldots,v_n}_{x_1,\ldots,x_n}P] = P(x_1, \ldots, x_n)$
2. $clf[(\Phi_1 \vee \Phi_2)] = (clf[\Phi_1] \vee clf[\Phi_2])$
3. $clf[\neg\Phi] = \neg clf[\Phi]$

Transformation *clf* preserves satisfiability.
Before proceeding to the proof we introduce several mappings over nominative sets and predicates related with a special constant ε. This constant represents the case when a value of a variable is not defined in a nominative set.
Let $A_\varepsilon = A \bigcup \{\varepsilon\}$, $\varepsilon \notin A$. First, define ε-adding transformation (ε-totalizing) *ade*: ${}^VA \xrightarrow{t} A^V_\varepsilon$ by the formula $ade(d)=[v \mapsto \varepsilon | \text{for each } v \in V] \nabla d$. In other words $ade(d) = [v \mapsto d(v) | v \in V,\ d(v)\downarrow] \nabla [v \mapsto \varepsilon | v \in V,\ d(v)\uparrow]$.
It is clear that *ade* is bijective. The inverse mapping will be denoted *dee*: $A^V_\varepsilon \xrightarrow{t} {}^VA$.
Formally, $dee(d) = [v \mapsto d(v) | v \in V,\ d(v) \neq \varepsilon]$.
In the same way we define two bijective mappings between classes of total predicates:
aep: $({}^VA \xrightarrow{t} Bool) \xrightarrow{t} (A^V_\varepsilon \xrightarrow{t} Bool)$ and *dep*:$(A^V_\varepsilon \xrightarrow{t} Bool) \xrightarrow{t} ({}^VA \xrightarrow{t} Bool)$.
For every $p \in (A^V_\varepsilon \xrightarrow{t} Bool)$, $dep(p)$ is defined as follows: given any $d \in {}^VA$ $dep(p)(d)=p(ade(d))$.
For every $p \in ({}^VA \xrightarrow{t} Bool)$, $aep(p)$ is defined as follows: given any $d \in A^V_\varepsilon$ $aep(p)(d)=p(dee(d))$.
From definitions we see that *dep*: $(A^V_\varepsilon \xrightarrow{t} Bool) \xrightarrow{t} ({}^VA \xrightarrow{t} Bool)$ is an isomorphism between algebras
$< (A^V_\varepsilon \xrightarrow{t} Bool), \vee, \neg, R^{\bar{v}}_{\bar{x}} >$ and $< ({}^VA \xrightarrow{t} Bool), \vee, \neg, R^{\bar{v}}_{\bar{x}} >$.

Theorem 7.1.

Let $\Phi \in Fr_{RCNL}(V, Ps)$. Then $|\approx_{RCNL} \Phi$ iff $|\approx_{FCL} clf[urnf[\Phi]]$.

Proof.

Let $\Phi_{CL} = clf[urnf[\Phi]]$, $J = ({}^VA, TPr({}^VA), I)$ be a π-interpretation, $\Phi_J(d_0)\downarrow = T$ for some $d_0 \in {}^VA$. We define J to interpret formulas as total predicates. This was justified by Lemma 5.4.

Now, given J, Φ, and d_0, we construct a classical interpretation $J_{CL} = (A_\varepsilon^V, TPr(A_\varepsilon^V),\ I_{CL})$. Interpretation I_{CL} for predicate symbols is defined as follows. Let $(v_1, \dots, v_n)$ be a list of upper names occurring in $urnf[\Phi]$. Then for every $P \in Ps$ we construct an n-ary predicate $p = I_{CL}(P)$ in the following way:

$$p(a_1, \dots, a_n) = I(P)([v \mapsto a | v \mapsto a \in_n d_0,\ v \notin \{v_1, \dots, v_n\}]\nabla[v_i \mapsto a_i | i = 1, \dots, n,\ a_i \neq \varepsilon]).$$

The interpretation J_{CL} induces interpretations of all sub-formulas of Φ_{CL} into predicate class $(A_\varepsilon^V \xrightarrow{t} Bool)$.
Let $D_\varepsilon = \{ade(d_0)\nabla d | d \in A_\varepsilon^{V_\Phi}\}$ and $D = \{dee(d) | d \in D_\varepsilon\}$. It is clear that ade as well as dee are bijections between D_ε and D.
We also have that clf is a bijection between sub-formulas of Φ and sub-formulas of Φ_{CL}. Now we prove by induction that any sub-formula Ψ of Φ and its counterpart $\Psi_{CL} = clf[\Psi]$ are interpreted as predicates that yield the same values on related (by ade) elements of D and D_ε respectively. That is: $\Psi_J(d) = (\Psi_{CL})_{J_{CL}}(ade(d)),\ d \in D$.
Base of induction: Let $\Psi = R^{v_1,\dots,v_n}_{x_1,\dots,x_n} P$, then $\Psi_{CL} = P(x_1, \dots, x_n)$. By construction of I_{CL} we have that $(R^{v_1,\dots,v_n}_{x_1,\dots,x_n} P)_J(d) = (P(x_1, \dots, x_n))_{J_{CL}}(ade(d))$, because $v_i \in V_\Phi$ $(i = 1, \dots, n)$. Inductive step is trivial.
Now from $\Phi_J(d_0)\downarrow = T$ we obtain that $(\Phi_{CL})_{J_{CL}}(ade(d_0))\downarrow = T$.
Let us prove the inverse. Suppose $(\Phi_{CL})_{J_{CL}}(d_\varepsilon) = T$ for some $d_\varepsilon \in A_\varepsilon^V$.
Let us construct an interpretation $J = ({}^V A, TPr({}^V A), I)$ and data d_0 such that $\Phi_J(d_0) = T$.
Let $D = \{dee(d) | d \in A_\varepsilon^{V_\Phi}\}$. For each P occurring in Φ within renominative atoms with $(v_1, \dots, v_n)$ as upper names and for every $d \in {}^V A$ we assign $I(P)(d) = I_{CL}(P)(ade(d)(v_1), \dots, ade(d)(v_n))$. Now we can prove in the same way that $\Phi_J(dee(d_\varepsilon)) = T$. □

Theorem 7.1 allows us to solve the satisfiability problem in RCNL by applying the methods for solving the same problem in classical quantifier-free predicate logic. Obtained formulas of the latter logic can further be transformed into equisatifiable formulas of pure propositional logic. However, this requires the data set A to have enough values to distinguish all $(x_1, \dots, x_n)$ tuples occurring in Φ_{CL}. Nevertheless, it is not important when we consider the general satisfiability problem, i.e., when we do not concretize the data set. Thus, on the renominative level we can reduce the satisfiability problem in CNL to the propositional satisfiability problem. The reduction is illustrated in the following examples.

Example 2.

Consider an RCNL formula Φ with three predicate symbols: P, Q, S. Let the set V contain x_1, y_1, x_2, y_2. Let

$$\Phi = R^{x_1}_{y_1}(P \vee Q) \wedge \neg R^{x_1,x_2}_{y_1,y_2}(P \vee R^{x_2,y_1}_{x_1,y_2} Q) \wedge S.$$

According to the procedure, we construct renominative normal form of Φ by applying simple transformations. Thus we obtain Φ_1.

$$\Phi_1 = (R^{x_1}_{y_1} P \vee R^{x_1}_{y_1} Q) \wedge \neg (R^{x_1,x_2}_{y_1,y_2} P \vee R^{x_1,x_2,y_1}_{y_1,y_1,y_2} Q) \wedge R\ S.$$

Now we construct URNF by unifying occurrences of the renominative atoms $R^{x_1}_{y_1} P$, $R^{x_1,x_2}_{y_1,y_2} P$, $R^{x_1}_{y_1} Q$, $R^{x_1,x_2,y_1}_{y_1,y_1,y_2} Q$, and $R\ S$. Thus we obtain Φ_2.

$$\Phi_2 = (R^{x_1,x_2,y_1}_{y_1,x2,y_1} P \vee R^{x_1,x_2,y_1}_{y_1,x_2,y_1} Q) \wedge \neg (R^{x_1,x_2,y_1}_{y_1,y_2,y_1} P \vee R^{x_1,x_2,y_1}_{y_1,y_1,y_2} Q) \wedge R^{x_1,x_2,y_1}_{x_1,x_2,y_1} S$$

Now we are ready to construct a classical predicate logic formula Φ_{CL} which has ternary predicates P, Q, S.

$$\Phi_{CL} = (P(y_1, x_2, y_1) \vee Q(y_1, x_2, y_1)) \wedge \neg (P(y_1, y_2,\ y_1) \vee Q(y_1, y_1, y_2)) \wedge S(x_1, x_2, y_1).$$

This formula is satisfiable. For example, we can consider the set of natural numbers as a set A. Then the latter formula is equisatisfiable with the following formula of propositional logic $(P_1 \vee Q_1) \wedge \neg\ (P_2 \vee Q_2) \wedge S$, which is clearly satisfiable. Thus we have established $|\approx \Phi$. Note that Φ is not satisfiable for data set $\{\alpha\}^V$. Indeed, if we used one-element set of

basic values then Φ_{CL} would be equisatisfiable with the propositional formula $(P_1 \vee Q_1) \wedge \neg (P_1 \vee Q_1) \wedge S$, which is not satisfiable.

Example 3.

Consider an RCNL formula Φ with two predicate symbols: P, Q. Let the set V contain x_1, y_1, x_2, y_2. Let

$$\Phi = R^{x_1,x_2}_{y_1,y_2}(P \wedge Q) \wedge \neg R^{x_2}_{y_2}(R^{x_1}_{y_1} P).$$

RNF of Φ is then

$$\Phi_1 = R^{x_1,x_2}_{y_1,y_2} P \wedge R^{x_1,x_2}_{y_1,y_2} Q \wedge \neg R^{x_1,x_2}_{y_1 y_2} P.$$

No unification of renominative atoms needed, so an equisatisfiable predicate logic formula Φ_{CL} has two binary predicates P and Q and is as follows:

$$\Phi_{CL} = P(y_1, y_2) \wedge Q(y_1, y_2) \wedge \neg P(y_1, y_2).$$

This formula is unsatisfiable in FCL, which means that Φ is unsatisfiable in RCNL.

8. Satisfiability for quantifier CNL

In this section we suggest a technique that allows reducing the satisfiability problem for CNL formulas of the quantifier level to the satisfiability problem for formulas of classical first-order predicate logic. This will be a more complicated reduction than in the case of renominative logic. The difficulties occur due to the composition of quantification which 1) in the general case is not distributive with renomination composition, and 2) does not have properties like "$(\forall x \Phi)_J(d) \downarrow = T$ implies $\Phi_J(d) \downarrow = T$" that hold in classical logic. To cope with these difficulties we should extend the language signature of composition-nominative logics with so-called unessential symbols upon which basic predicates do not depend, and for classical logic additionally add a symbol of unary predicate of equality to a constant. This will allow us to construct URNF for a given QCNL formula, and to transform it then to an equisatisfiable QCL formula in the extended language. Therefore, let us generalize the rules for constructing *rnf* and *urnf* to extend these mappings onto $Fr_{QCNL}(V, Ps)$.
The rules for constructing RNF can be extended as follows.
R8) $rnf[\exists y\ \Phi] = \exists y\ rnf[\Phi]$;
R9a) $rnf[R^{\bar{v}}_{\bar{x}}\ \exists y\ \Phi] = \exists y\ rnf[R^{\bar{v}}_{\bar{x}} \Phi]$, when y does not occur in $\bar{v}$ and $\bar{x}$;
R9b) $rnf[R^{\bar{v}}_{\bar{x}}\ \exists y\ \Phi] = \exists u\ rnf[R^{\bar{v}}_{\bar{x}}\ R^{y}_{u}\ \Phi]$, when y occurs in $\bar{v}$ or $\bar{x}$. Here u is a fresh unessential variable, $u \in U$ (see below). If $\bar{v}$ and $\bar{x}$ are empty lists, rule R9b represents the rule of quantified variable renaming.
Note that each application of the R9b rule introduces an unessential variable u. This variable belongs to a set of unessential variables U such that $U \cap V = \emptyset$, U is infinite. Unessential variables from U put a restriction on any σ-interpretation I^{Ps}: $Ps \xrightarrow{t} Pr^{V \cup U, A}$ in such a way that for every $P \in Ps$ and any $d \in {}^{V \cup U}\mathrm{A}$ the value of $I^{Ps}(P)(d)$ does not depend on values of variables from the set U. Actually it means that we consider a logic with a restricted class of interpretations and the extended language $Fr_{QCNL}(V \cup U, Ps)$. By a *fresh variable* for a formula we mean a variable that does not occur in the formula. The rule R9b also permits to assume w.l.o.g. that all quantified variables in the formula are different.
The notion of URNF has to be adjusted for QCNL. An additional requirement is that for every renominative atom $R^{\bar{v}}_{\bar{x}} P$, and every quantifier $\exists y$ that occur in the initial formula Φ, y shall occur in $\bar{v}$. During URNF construction we ensure fulfillment of this requirement by additional applications of the identical renomination rule. Note that at least one quantifier shall occur in the formula; otherwise we get the formula of the levels already discussed.
Following the definition of transformation $clf : Fr_{RCNL}(V, Ps) \xrightarrow{p} Fr_{FCL}(V, Ps)$ let us extend the reduction to $clf : Fr_{QCNL}(V, Ps) \xrightarrow{p} Fr_{QCL}(V \bigcup U, Ps)$ which maps every QCNL formula in URNF to a QCL formula that represents the same term up to atomic sub-formulas.

1. $clf[R^{v_1,\dots,v_n}_{x_1,\dots,x_n} P] = P(x_1, \dots, x_n)$
2. $clf[(\Phi_1 \vee \Phi_2)] = (\, clf[\Phi_1] \vee clf[\Phi_2])$
3. $clf[\neg\Phi] = \neg clf[\Phi]$
4. $clf[\exists x\ \Phi] = \exists x\ clf[\Phi]$

A reasonable question one may ask would be whether we now have that $|\approx_{QCNL} \Phi \Leftrightarrow |\approx_{QCL} clf[urnf[\Phi]]$. The answer is illustrated by the counterexample below.

Example 4.

Consider a QCNL formula

$$\Phi = (\forall x \neg P) \wedge P.$$

It is easy to show that this formula is satisfiable in QCNL. Indeed, let $J = ({}^{V}A,\ Pr^{V,A},\ I)$ be such that $V=\{x,\ y\}$, $A=\{1,\ 2\}$. Let $I(P)(d)\downarrow = F$ if $x \mapsto a \in_n d$ for some $a \in A$ and T in all other cases. In other words, the predicate P takes the value T on some data d iff the name x is undefined in d. Note that the predicate $I(P)$ is not equitone. Indeed, $I(P)([x \mapsto 1, y \mapsto 1])\downarrow = F$, whereas $I(P)([y \mapsto 1])\downarrow = T$. Now we have that $\Phi_J\ ([y \mapsto 1])\downarrow = T$, which means that Φ is satisfiable. At the same time it is easy to prove that the formula

$$clf[urnf[\Phi]] = (\forall x \neg P(x)) \wedge P(x),$$

is not satisfiable in the classical first-order predicate logic. This example also demonstrates that under *dep*/*aep* mappings algebras $< (A^V_\varepsilon \xrightarrow{t} Bool),\ \vee, \neg\ ,\ R^{\bar{v}}_{\bar{x}},\ \exists x >$ and $< ({}^{V}A \xrightarrow{t} Bool),\ \vee, \neg\ ,\ R^{\bar{v}}_{\bar{x}},\ \exists x >$ are not isomorphic.

To reduce the satisfiabilty problem in QCNL to the satisfiability problem in QCL we need to refine the formula transformation to classical logic (this concerns the rule 4). It also requires us to introduce additional unary predicate symbol into the signature of QCL, so, terms representing transformed formulas will have a more complicated structure. The idea is to get the possibility to check whether a variable has a value (is defined). This can be done with the help of a unary predicate of equality to constant ε denoted $\varepsilon{=}$. Atomic formulas for this unary symbol have the form $\varepsilon{=}(x)$. For simplicity's sake we write $x \neq \varepsilon$ instead of $\neg(\varepsilon{=}(x))$. Let $PsE{=}Ps{\cup}\{\varepsilon{=}\}$. Consequently, we get a new extended predicate algebra.
We formalize the syntactical reduction clf of QCNL formulas in unified renominative normal form to QCL formulas changing the rule 4 to the following new rule:

4. $clf[\exists x\Phi] = \exists x(x \neq \varepsilon \wedge clf[\Phi])$

Lemma 8.1.

Let $\Phi \in Fr_{QCNL}(V{\cup}U,\ Ps)$ be in unified renominative normal form, $J = ({}^{V\cup U}A,\ TPr({}^{V\cup U}A),\ I^{\,Ps})$ be a π-interpretation such that $J\ |\approx \Phi$, $\Phi_{CL} \in Fr_{QCL}(V{\cup}U,\ PsE)$ be such that $\Phi_{CL}{=}clf[\Phi]$. Then there exists a κ-interpretation $J_{CL} = (A^{V\cup U}_\varepsilon,\ TPr(A^{V\cup U}_\varepsilon),\ I^{PsE}_{CL})$ such that $J_{CL}|\approx \Phi_{CL}$.

Proof.

With minor modifications the proof follows the proof of Theorem 7.1.
For convenience let us recall our constructions. Let $(v_1, \dots, v_n)$ be the list of upper names occurring in renominative atoms of $urnf[\Phi]$, $V_\Phi = \{v_i | i = 1, \dots, n\}$. Let $\Phi_J(d_0)\downarrow = T$ for some $d_0 \in {}^V A$. We should construct a classical interpretation $J_{CL} = ((A \bigcup \{\varepsilon\})^{V \bigcup U}, TPr(A_\varepsilon^{V \bigcup U}), I_{CL}^{PsE})$. Then for every $P \in Ps$ we define an n-ary predicate $p = I_{CL}^{PsE}(P)$ as follows: $p(a_1, \dots, a_n) = I^{Ps}(P)([v \mapsto a | v \mapsto a \in_n d_0,\ v \notin \{v_1, \dots, v_n\}]\nabla[v_i \mapsto a_i | i = 1, \dots, n,\ a_i \neq \varepsilon])$.
Let $D_\varepsilon = \{ade(d_0)\nabla d | d \in A_\varepsilon^{V_\Phi}\}$ and $D = \{dee(d) | d \in D_\varepsilon\}$.
The induction statement that needs proving is that any sub-formula Ψ of Φ and its counterpart $\Psi_{CL} = clf[\Psi]$ are interpreted as predicates that yield the same values on related (by *ade*) elements of D and D_ε respectively. That is: $\Psi_J(d) = (\Psi_{CL})_{J_{CL}}(ade(d))$, $d \in D$.
We prove the induction statement only for the case of existential quantifier.
Let $\Psi = \exists x \Theta$, then $\Psi_{CL} = clf[\Psi] = \exists x(x \neq \varepsilon \wedge \Theta_{CL})$, where $\Theta_{CL} = clf[\Theta]$. By induction base we have that for every $d \in D$ $\Theta_J(d) = (\Theta_{CL})_{J_{CL}}(ade(d))$. Let us prove that for every $d \in D$ $\Psi_J(d) = (\Psi_{CL})_{J_{CL}}(ade(d))$.

1. Suppose $\Psi_J(d) = T$. That is, $(\exists x\ \Theta)_J(d) = T$. It means there is some $a \in A$ such that $\Theta_J(d\nabla x \mapsto a) = T$. As $x \in V_\Phi$ we have that $d \in D \Rightarrow d\nabla x \mapsto a \in D$. That means, according to induction assumption, that $(\Theta_{CL})_{J_{CL}}(ade(d\nabla x \mapsto a)) = T$. Note that $ade(d\nabla x \mapsto a) = ade(d)\nabla x \mapsto a$. That is, $(\Theta_{CL})_{J_{CL}}(ade(d)\nabla x \mapsto a) = T$. Consequently, $(\exists x(x \neq \varepsilon \wedge \Theta_{CL}))_{J_{CL}}(ade(d)) = T$. So we have obtained that $\Psi_J(d) = T \Rightarrow (\Psi_{CL})_{J_{CL}}(ade(d)) = T$.

2. Suppose that $\Psi_J(d) = F$. That is, $(\exists x\ \Theta)_J(d) = F$. Then we have that for all $a \in A$ $\Theta_J(d\nabla x \mapsto a) = F$. That means, according to induction assumption, that for all $a \in A$ $(\Theta_{CL})_{J_{CL}}(ade(d\nabla x \mapsto a)) = F$. Which means that for all $a \in A$ $(\Theta_{CL})_{J_{CL}}(ade(d)\nabla x \mapsto a) = F$. Let us assume that $(\Theta_{CL})_{J_{CL}}(ade(d)\nabla x \mapsto b) = T$ for some $b \in A_\varepsilon = A \bigcup \{\varepsilon\}$. It then follows immediately that $b = \varepsilon$. That means that $(x \neq \varepsilon \wedge \Theta_{CL})_{J_{CL}}(ade(d)\nabla x \mapsto b) = F$ for all $b \in A_\varepsilon$. Hence $(\exists x(x \neq \varepsilon \wedge \Theta_{CL}))_{J_{CL}}(ade(d)) = F$. Thus $\Psi_J(d) = F \Rightarrow (\Psi_{CL})_{J_{CL}}(ade(d)) = F$.

As I^{Ps} assigns to each predicate symbol a total interpretation, the value of $\Psi_J(d)$ is always defined (Lemma 5.5). Thus we can say that for every $d \in D$ $\Psi_J(d) = (\Psi_{CL})_{J_{CL}}(ade(d))$.
The rest of the proof is identical to that of Theorem 7.1. □

Theorem 8.1.

Let $\Phi \in Fr_{QCNL}(V, Ps)$. Then $|\approx_{QCNL} \Phi$ iff $|\approx_{QCL} clf[urnf[\Phi]]$.

Proof.

Due to Lemma 5.4 we can consider only interpretations J with total predicates. The direct statement follows from Lemma 8.1, the inverse statement is proved in the same way as described in the proof of Theorem 7.1. □

Theorem 8.1 justifies our method for checking satisfiability of formulas in composition-nominative logics, which consists in transforming them to equisatisfiable formulas of classical predicate logic with extended language.
Let us illustrate this method on the formula from example 4.

Example 5.

Consider a QCNL formula

$$\Phi = (\forall x \neg P) \wedge P.$$

A unified renominative normal form of Φ is

$$\Phi_1 = (\forall x \neg R^x_x P) \wedge R^x_x P.$$

Therefore

$$\Phi_{CL} = clf[\Phi_1] = (\forall x(x \neq \varepsilon \rightarrow \neg P(x))) \wedge P(x).$$

It is easy to see that the first-order formula Φ_{CL} is satisfiable. Indeed, let $J_{CL} = (A_\varepsilon^{V \bigcup U}, TPr(A_\varepsilon^{V \bigcup U}),\ I_{CL}^{PsE})$ be such that $A_\varepsilon = A \bigcup \{\varepsilon\}$, $A = \{0\}$, $V = \{x\}$, $I(P)([x \mapsto 0]) = F$, $I(P)([x \mapsto \varepsilon]) = T$. Then we get that $(\Phi_{CL})_{J_{CL}}([x \mapsto \varepsilon]) = T$, which means, according to Theorem 8.1, that Φ is satisfiable. One possible interpretation J such that $J \mid\approx \Phi$ was provided at the end of example 4.

9. Related work

The main aspects of the composition-nominative approach such as partiality, compositionality, nominativity, validity, have received a lot of attention and have been investigated for decades. This is reflected in works in such fields as logic, philosophy, linguistics, and computer science.

The importance of partiality, for example, was already being discussed in detail by the 1980's [6]. Since that time many different approaches have emerged. A survey of some of such approaches and a comparison of several formalisms can be found in [11, 24]. Partiality continues to receive more and more attention in the computer science community. Theorem proving systems [16] and validity checkers [4] tend to introduce support also for partial functions.

Compositionality can be traced back to works of Gottlob Frege. The history of this principle can be found in [10]. Nowadays the importance of the compositionality principle increases because of the necessity of investigation and verification of complex systems [5, 26], in particular, concurrent systems [7]. In our approach we take compositionality as a basic principle. When we consider functions (predicates) as meanings of expressions (of formulas) the constructed formal languages are compositional by construction.

Nominativity is another fundamental aspect not only in computer science but also in other branches of science, especially in linguistics and philosophy. This topic requires a special consideration, but still we would like to outline the nominal logic [25], which shares some similarities with the logics defined in this paper. Nominal logic addresses several special questions of nominativity such as name bindings, freshness and swapping. The predicates considered in the nominal logic have to be equivariant. That is, their validity should be invariant under name swapping. In our approach we consider general classes of partial predicates.

There are also different approaches for treating validity in partial logics. A problem, which is relevant to the one discussed here has been studied in [4]. In this work the validity problem in the three-valued logic is addressed. The approach proposed in [4] and the approach discussed here both aim at reducing the problem of checking validity (satisfiability) to the validity (satisfiability) problem in classical logic. However, the details of two approaches are different. In the work mentioned, the authors consider n-ary functions and a three-valued notion of validity. That is, a formula can be proved to be "valid", "invalid", or its validity can be "undefined". A formula that evaluates to true in all interpretations under all variable assignments is called valid. If there is at least one interpretation and one variable assignment such that the formula evaluates to false - it is called invalid. In other cases the validity for the formula is undefined. An additional restriction is that functions and predicates are required to have explicit specifications of their domains. These specifications are represented by formulas in the two-valued logic. These formulas allow establishing a condition under which the original formula is always defined. If such a condition is valid, the formula is checked for validity in the two-valued logic. We find these ideas interesting and plan to apply them for logics of quasiary predicates.

To the best of our knowledge, the satisfiability problem in logics similar to composition-nominative logics has not been previously addressed. An in-depth comparison of composition-nominative approach with other approaches addressing compositionality, nominativity, validity or permitting reasoning about partial functions and predicates is certainly beyond the scope of this paper. However we would like to stress on several important features. Our approach is based on algebras of partial predicates over nominative data, in particular, on algebras of quasiary functions and predicates as opposed to more conventional algebras of n-ary functions and predicates. It involves new compositions, among them the

renomination composition. These compositions take into account nominative aspects of data structures. Composition-nominative approach also advocates the semantic-syntactic style of logic definitions, which simplifies the construction and investigation of such logics.

10. Conclusions

This paper is essentially a first step in investigating the satisfiability problem for composition-nominative logics, which are algebra-based logics of partial predicates that do not have fixed arity. As a main result we have proved that the satisfiability problem for all types of composition-nominative logics considered here can be reduced to the same problem for classical predicate logic with more powerful language. Thus, existent state-of-the-art methods and techniques for checking satisfiability in classical logic can also be applied to CNL.

We have defined and investigated the satisfiability problem for three different levels of CNL: propositional, renominative, and quantifier levels. For propositional CNL we have proved that satisfiability of formulas can be checked in classical propositional logic. For CNL of the renominative level we have proposed a method of reduction of the satisfiability problem to the same problem for quantifier-free classical predicate logic. For quantifier level of CNL we have reduced the satisfiability problem to the satisfiability problem for the classical first-order predicate logic. The latter reduction requires extension of the language with unessential variables and an interpreted unary predicate of equality to a constant. The interpretation domain of classical logic should also be extended with a special value to reflect possible undefinedness of nominative data components in CNL.

One direction for future work includes investigation of satisfiability problem for more powerful CNL of predicate-function level and for CNL over hierarchic nominative data. Hierarchic data allow the representation of such complex structures as lists, stacks, arrays etc.; thus, such logics will be closer to program models with richer data types. Another direction is related with identification of classes of formulas in various types of CNL for which satisfiability problem can be solved efficiently. In particular, this concerns specialized theories, when some predicates have specific interpretations and several axioms shall hold for such interpretations. This is often referred to as the satisfiability modulo theory (SMT) problem [2, 19]. The SMT problem occurs in practice in the areas of software/hardware verification, program analysis, testing, etc. For some logical theories, for example, linear integer arithmetic (Presburger arithmetic) this problem is decidable, but for the most theories of practical interest the SMT problem is decidable only for their quantifier-free fragments [13]. Lastly, prototypes of software systems for satisfiability checking in CNL should be developed.

Acknowledgements

The authors would like to thank anonymous reviewers for reading early versions of this paper and providing helpful feedback.

References

[1] Abrial J.-R.,The B-Book: Assigning Programs to Meanings, Cambridge University Press, 1996

[2] Barrett C., Sebastiani R, Seshia S. A., Tinelli C., Satisfiability Modulo Theories, In: Biere A., Heule M., van Maaren H., Walsh T. (Eds.), Handbook of Satisfiability, IOS Press, 2009

[3] Basarab I.A., Gubsky B.V., Nikitchenko N.S., Red'ko V.N., Composition Models of Databases, In: Eder J., Kalinichenko L.A. (Eds.), East-West Database Workshop (Workshops in Computing Series), 221-231, Springer, London, 1995

[4] Berezin S. et al., A Practical Approach to Partial Functions in CVC Lite, Electron. Notes Theor. Comput. Sci., 125, 13-23, 2005

[5] Bjørner D., Eir A., Compositionality: Ontology and Mereology of Domains, Lect. Notes Comput. Sc., 5930, 22-59, 2010

[6] Blamey, S., Partial Logic, In: Gabbay D., Guenthner F. (Eds.), Handbook of Philosophical Logic, Volume III, D. Reidel Publishing Company, Dordrecht, 1986
[7] Dams D., Hannemann U., Steffen M. (Eds.), Concurrency, Compositionality, and Correctness, Essays in Honor of Willem-Paul de Roever, Lect. Notes Comput. Sc., 5930, Springer, Heidelberg, 2010
[8] Davis M., Logemann G., Loveland D., A machine program for theorem-proving, COMMUN ACM., 5(7), 394-397, 1962
[9] Gu J., Purdom P.W., Franco J., Wah B. W., Algorithms for the Satisfiability (SAT) Problem: A Survey, In: Du D., Gu J., Pardalos P.M. (Eds.), Satisfiability Problem: Theory and Applications, DIMACS SER DISCRET M., 35, 19-152, 1997
[10] Janssen T.M.V., Compositionality, In: van Benthem J., ter Meulen A. (Eds.), Handbook of Logic and Language, Elsevier and MIT Press, Amsterdam/Cambridge, 1997
[11] Jones C.B., Reasoning About Partial Functions in the Formal Development of Programs, Electron. Notes Theor. Comput. Sci., 145, 3-25, 2006
[12] Kleene, S.C.: Introduction to Metamathematics, Van Nostrand, New York, 1952
[13] Kroening D., Strichman O., Decision Procedures - an Algorithmic Point of View, Springer-Verlag, Berlin Heidelberg, 2008
[14] Lamport L., Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers, Addison-Wesley, 2002
[15] Marques-Silva J., Practical Applications of Boolean Satisfiability, In: Workshop on Discrete Event Systems (28-30 May 2008, Goteborg, Sweden), 74-80
[16] Mehta F. A., Practical Approach to Partiality. A Proof Based Approach, Lect. Notes Comput. Sc., 5256, 238-257, 2005
[17] Mendelson E., Introduction to Mathematical Logic, 4th ed., Chapman & Hall, London, 1997
[18] Nielson H.R., Nielson F., Semantics with Applications: A Formal Introduction. John Wiley & Sons Inc, 1992
[19] Nieuwenhuis R., Oliveras A., Tinelli C., Solving SAT and SAT modulo theories: from an abstract Davis-Putnam-Logemann-Loveland procedure to DPLL(T), J ACM., 53, 937-977, 2006
[20] Nikitchenko, N.S.: A Composition Nominative Approach to Program Semantics. Technical Report IT-TR 1998-020, Technical University of Denmark, 103, 1998
[21] Nikitchenko N.S., Abstract computability of non-deterministic programs over various data structures, In: Bjørner D., Broy M., Zamulin A.V. (Eds.), Perspectives of System Informatics, Lect. Notes Comput. Sc., 2244, 471-484, 2001
[22] Nikitchenko M.S., Composition-nominative aspects of address programming, Kibernetika I Sistemnyi Analiz, 6, 24-35, 2009 (In Russian)
[23] Nikitchenko M.S., Shkilnyak S.S., Mathematical logic and theory of algorithms, Publishing house of Taras Shevchenko National University of Kyiv, Kyiv, 2008 (In Ukrainian)
[24] Owe O., Partial Logics Reconsidered: A Conservative Approach, FORM ASP COMPUT., 5, 208-223, 1997
[25] Pitts, A. M., Nominal Logic, A First Order Theory of Names and Binding, INFORM COMPUT., 186, 165-193, 2003
[26] de Roever, W.-P., Langmaack H., Pnueli A. (Eds.), Compositionality: The Significant Difference, Lect. Notes Comput. Sc., 1536, VIII, Springer, Heidelberg, 1998
[27] Schmidt D.A., Denotational Semantics: A Methodology for Language Development, Allyn and Bacon, 1986
[28] Shkilniak S. S., First-order logics of quasiary predicates, Kibernetika I Sistemnyi Analiz, 6, 32-50, 2010 (In Russian)
[29] Spivey J.M., The Z Notation: A Reference Manual, 2nd ed., Prentice Hall, 1992
[30] Winskel G., The Formal Semantics of Programming Languages, MIT Press, 1993