Nondeterministic Algebraic Specifications in Relational Syntax

Valentinas Kriaučiukas [1] *Sigurd Meldal* [2] *Michał Walicki* [2]

[1]Matematikos ir Informatikos Institutas, Akademijos 4, 2600 Vilnius, Lithuania
`valentinas.kriauciukas@mlats.mii.lt`

[2]Institutt for Informatikk, Universitetet i Bergen, HiB, N-5020 Bergen, Norway
`{sigurd,michal}@ii.uib.no`

Abstract

Relational syntax is presented for nondeterministic algebraic specifications called the language of relational terms. It is shown how nondeterministic specifications can be translated to the relational terms.

A complete inference system is presented for derivation of valid clauses of relational atoms.

Finally, a λ-abstraction construct is introduced extending the syntax of relational terms. It is weaker than the functional λ-abstraction, but allows translation of first-order predicate language into the language of relational terms. A slight extension of the inference system remains sound and complete for this extended syntax.

Introduction

The paper presents work in progress on establishing connections between the theory of nondeterministic specifications and other relevant theories, in particular, relational algebra [ABHVW], λ-calculus and first-order predicate calculus. In our earlier work [KW95, WM95a, WM95b] nondeterministic specifications are sets of clauses, where literals are written using the binary predicates $\prec$ (inclusion), $\approx$ (deterministic equality), $\frown$ (non-emptiness of intersection) and their negated forms. The term syntax is as usual, but semantically function symbols are interpreted as *nondeterministic operators*, *i.e.*, returning possibly more than one value for a given argument value. In fact, nondeterministic operators are *oriented* relations, *i.e.*, for a relation there is a distinguished argument representing the possible return values of the corresponding operator.

We introduce a language of *relational terms* which masquerades as a first-order language of relational algebra. We obtain a complete axiomatization of validity of clauses of relational terms. On the other hand, we have earlier introduced [KW95] complete system for reasoning about these specific relations. The system is prone to automation and we hope that, having established the connections with relational algebra, it (or its variant) could also be applied in the relational setting.

The language of relational terms explicitly uses composition, product, complement and inverse of binary relations, and can also be extended with union (choice) and λ-abstraction. The interesting feature of the language is that each term denotes a binary relation (except variables, which denote only specific relations, namely, individual elements). Thus a term is a kind of statement, and thus we have a *purely logical* (and algebraic, since there are no quantifiers) language, similar to that of propositional calculus. This level of "purity" is the same as in the language of λ-calculus, where each term denotes a function. Differences and similarities with λ-calculus can be studied by explicit introduction of λ-abstraction, This opens a wide field of interesting investigations of how one may use relational terms wherever the λ-calculus is used. The language presented here does not explore the full power of λ-abstraction because variables denote only individuals but not arbitrary relations. The construct as it is used here serves the purpose of rearranging the arguments of relations.

The composition of relations and its dual, both expressing implicit quantification, allow direct encoding of the language of first-order predicate calculus in our (quantifier-free) language of relational terms, without any skolemisation, *i.e.*, without the introduction of additional function symbols.

In Section 1 we introduce nondeterministic algebraic specifications as in [KW94, KW95]. Then, in Section 2, the language of relational terms is defined. A complete axiomatic system for this language is presented in Section 3, the completeness proof of which is moved into Appendix A. Section 4 is devoted to extension of relational terms with λ-abstraction, which enables simple translation of the language of first order predicates into syntax of relational terms.

1 Language of nondeterministic specifications

1.1 Syntax

Language of nondeterministic algebraic specifications is not firmly fixed, the main thing to vary is the set of predicates. Here we start with one of our variants [KW94, KW95] containing three binary predicates. Specifications are written using a countable set of variables $\mathcal{V}$ and a finite non-empty set of functional symbols Σ, each $f \in \Sigma$ has a fixed arity $ar(f) \in \mathbb{N}$. A functional symbol c with $ar(c) = 0$ is called a *constant*. The set $\mathcal{T}(\Sigma, \mathcal{V})$ of terms of signature Σ is defined in usual way. There are three atomic forms of formulae built using binary predicates: *equation* $s \approx t$, *inclusion* $s \prec t$ and *intersection* $s \frown t$, connecting terms s and t. Atoms and their negations form the set of *literals*. An atom a is called a *positive* literal, and a negated atom $\neg a$ is a *negative* literal. Negative literals are written using forms $s \not\approx t$, $s \not\prec t$ and $s \not\frown t$. By $\overleftarrow{s \oplus t}$ the reversed literal $t \overleftarrow{\oplus} s$ is denoted, which is different from $t \oplus s$ only in the cases $\oplus \in \{\prec, \not\prec, \succ, \not\succ\}$ because these signs are not symmetric (by their shape and meaning).

A *specification* is a set of *clauses*, where each clause is a finite set of literals (*i.e.*, multiplicity and ordering of the atoms do not matter). By *words* we will mean the union of the sets of terms, literals and clauses.

1.2 Semantics

Syntactic expressions of the language are interpretated in *multialgebras* [Kap88, Hus92, WM95b] which, unlike usual algebras, allow functions to have multiple values. Here we also allow them to be *partial*, *i.e.*, do not have any value for some arguments.

Definition 1.1 *A* Σ-multialgebra A *is a tuple* $\langle S^A, \Sigma^A \rangle$, *where* S^A *is a* carrier, *and* Σ^A *is a set of set-valued functions* $f^A : (S^A)^n \to \mathcal{P}(S^A)$ *where* $f \in \Sigma$, $n = ar(f)$ *and* $\mathcal{P}(S^A)$ *is the power-set of* S^A.

In definition of meaning of words we follow [WM95b, WM95a]. We denote boolean values *true* by T and *false* by F.

Definition 1.2 *Let* A *be a* Σ*-multialgebra,* $\varepsilon : \mathcal{V} \to S^A$ *be an interpretation of variables, then value* $[\![w]\!]_\varepsilon$ *of a word* w *is a subset of* S^A *and is defined as follows:*

1. *for a variable* $v \in \mathcal{V}$, $[\![v]\!]_\varepsilon \stackrel{\text{def}}{=} \{\varepsilon(v)\}$;
2. *for a constant* $c \in \Sigma$, $[\![c]\!]_\varepsilon \stackrel{\text{def}}{=} c^A$;
3. *for a term* $t = f(t_1, \ldots, t_n) \in \mathcal{T}$, $[\![t]\!]_\varepsilon \stackrel{\text{def}}{=} \bigcup_{\vec{\alpha} \in [\![\vec{t}]\!]_\varepsilon} f^A(\vec{\alpha})$, *where* $[\![\vec{t}]\!]_\varepsilon \stackrel{\text{def}}{=} [\![t_1]\!]_\varepsilon \times \cdots \times [\![t_n]\!]_\varepsilon$;
4. *for a literal* $l = s \oplus t$, $[\![l]\!]_\varepsilon \stackrel{\text{def}}{=} F_\oplus([\![s]\!]_\varepsilon, [\![t]\!]_\varepsilon)$, *where*
$$\begin{array}{rclrcl} F_\approx(X,Y) & \equiv & \forall \alpha \in X\ \forall \beta \in Y\ \alpha = \beta, & F_\prec(X,Y) & \equiv & X \subseteq Y, \\ F_\frown(X,Y) & \equiv & X \cap Y \neq \emptyset, & F_{\neg\otimes}(X,Y) & \equiv & \neg F_\otimes(X,Y); \end{array}$$
5. *for a clause* C, $[\![C]\!]_\varepsilon \stackrel{\text{def}}{=} \bigvee_{l \in C} [\![l]\!]_\varepsilon$, *if* $C \neq \emptyset$ *and* $[\![C]\!]_\varepsilon \stackrel{\text{def}}{=} \mathsf{F}$, *otherwise.*

Definition of equality $\approx$ is not perfect, because it makes empty set equal to any other set. In our earlier works, like [WM95b, KW94], only total functions were considered: the value sets of terms were always nonempty, and the definition worked well. In the case of partial functions applications of equality should be guarded somehow if one wants to focus on and exclude this anomaly.

Definition 1.3 *A Σ-multialgebra A* satisfies *a clause C if $[\![C]\!]_\varepsilon = \mathsf{T}$ for every interpretation of variables ε.*

A satisfies *a literal l iff it satisfies the clause $\{l\}$, and* satisfies *a specification* Spec *iff it satisfies all clauses in* Spec.

2 Language of relational terms

2.1 Syntax

We start with the same basic things as in the case of nondeterministic specifications: a countable set of variables $\mathcal{V}$ and a finite non-empty set of functional symbols Σ. We allow now functional symbols to have results of any arity, not only equal one, therefore, for each $f \in \Sigma$, the arity of results is $re(f) \in \mathbb{N}$. Symbols with $re(p) = 0$ are *predicate* symbols, while the ones with $ar(c) = 0$ are (multidimensional) *constants*. We write the full profile of a functional symbol f in the form $ar(f) \to re(f)$. The output dimension of functional symbols and, consequently, of terms is important for the definition of terms, which now are called *relational terms*. Since it is cumbersome to indicate this explicitly for each functional symbol, we extend the arity functions ar and re to the set of all terms. The way symbols are composed is now different from that described in Section 1. Variables are not relational terms and do not range over arbitrary sets – we are restricting the language to the first-order level, where variables range only over individual elements. In the following definition, $\dot{-}$ is a restricted substraction: $n \dot{-} 1$ is equal $n - 1$ if $n > 0$, and 0, otherwise.

Definition 2.1 *Let $\mathcal{V}$ be a set of variables, Σ be a set of functional symbols. The set $\mathcal{RT}(\Sigma, \mathcal{V})$ of* relational terms *and their arities are defined inductively as follows:*

1. T *and* F *are relational terms of arity* $0 \to 0$*;*
2. *I is a relational terms of arity* $1 \to 1$*;*
3. *each functional symbol $f \in \Sigma$ is a relational term;*
4. *for a variable $x \in \mathcal{V}$ and a relational term t, $x.t$ and $t.x$ are relational terms of arities $ar(t) \dot{-} 1 \to re(t)$ and $ar(t) \to re(t) \dot{-} 1$, respectively;*
5. *for any relational term t, there are relational terms $\neg t$ of the same arity as t, and $\overleftarrow{t}$ of arity $re(t) \to ar(t)$;*

6. *for any two relational terms s, t there are relational terms $s \times t$ and $s + t$ of arity $ar(s) + ar(t) \to re(s) + re(t)$;*
7. *for relational terms s and t with $ar(t) = re(s)$, there are relational terms $s \circ t$ and $s \diamond t$ of arity $ar(s) \to re(t)$;*
8. *for relational terms s and t, both of an arity $n \to m$, there are relational terms $s \sqcup t$ and $s \sqcap t$ of the same arity $n \to m$.*

The terms of arity $0 \to 0$ are called *atoms*, finite sets of atoms are *clauses*, and sets of clauses form *specifications*.

2.2 Semantics

Models where relational terms are interpreted we also call *multialgebras*, but in the presence of reversal $\overleftarrow{t}$ and complement $\neg t$ it is easier to deal with binary relations instead of set-valued functions. Like in the case of functions we write $R : D \to C$ to denote the fact $R \subseteq D \times C$, then D is called a *domain* and C is a *codomain* of R. The complement of R, $(D \times C) \setminus R$, is denoted R^{c}.

Definition 2.2 *A* Σ-multialgebra *A is a tuple $\langle S^A, \Sigma^A \rangle$, where S^A is a* carrier *set and Σ^A is a set of binary relations $\{f^A : (S^A)^{ar(f)} \to (S^A)^{re(f)} : f \in \Sigma\}$.*

Elements of a set $(S^A)^n$ are sequences of the length n and $(S^A)^0$ is the set containing only the empty sequence Λ. Relations of type $\Lambda \to \Lambda$ form very special case: they are subsets of the set $(S^A)^0 \times (S^A)^0 \cong (S^A)^0$. Obviously, there are only two different relations of this type. In this way we obtain very natural introduction of boolean values.

For two binary relations R and Q having as domains and codomains sets of sequences, we denote

$$R \times Q \stackrel{\text{def}}{=} \{\langle xu, yv\rangle : \langle x, y\rangle \in R \wedge \langle u, v\rangle \in Q\}$$

product of relations R, Q, and

$$R + S \stackrel{\text{def}}{=} \{\langle xu, yv\rangle : \langle x, y\rangle \in R \vee \langle u, v\rangle \in Q\}$$

some kind of sum, which is dual (not in a categorical sense) to the product: $R + Q = (R^{\mathsf{c}} \times Q^{\mathsf{c}})^{\mathsf{c}}$. The composition of the same relations, assuming that the domain of Q is the same as the codomain of R, is denoted

$$R \circ Q \stackrel{\text{def}}{=} \{\langle x, y\rangle : \exists z(\langle x, z\rangle \in R \wedge \langle z, y\rangle \in Q)\},$$

while the operation dual to composition is called *relational sum* [Pr88]. We call it *co-composition* and, instead of the strange notation used in [Pr88] (+ with "scorpion tail"), denote it by

$$R \diamond Q \stackrel{\text{def}}{=} (R^{\mathsf{c}} \circ Q^{\mathsf{c}})^{\mathsf{c}}.$$

The reverse of the relation R is denoted by $\overleftarrow{R}$ and not only interchanges arguments, but also reverses them as sequences:

$$\overleftarrow{R} \stackrel{\text{def}}{=} \{\langle \overleftarrow{y}, \overleftarrow{x} \rangle : \langle x, y \rangle \in R\},$$

where $\overleftarrow{y}$, $\overleftarrow{x}$ denote reversed sequences y and x. This not very usual kind of reversal is necessary to obtain the property $\overleftarrow{x.t} = \overleftarrow{t}.x$ (the last argument becomes the first result).

We also need "cuts" of relations: if $R : (S^A)^n \to (S^A)^m$ is some relation, and $a \in S^A$ some carrier element, then

$$aR \stackrel{\text{def}}{=} \begin{cases} \{\langle x, y \rangle : \langle xa, y \rangle \in R\}, & \text{if } n > 0, \\ R, & \text{if } n = 0 \end{cases}$$

and

$$Ra \stackrel{\text{def}}{=} \begin{cases} \{\langle x, y \rangle : \langle x, ay \rangle \in R\}, & \text{if } m > 0, \\ R, & \text{if } m = 0 \end{cases}$$

are two cuts of R by some hyperplanes. The identity relation on a set S, $\{\langle a, a \rangle : a \in S\}$, is denoted $\boldsymbol{I}_S$.

Definition 2.3 *Let A be a Σ-multialgebra, $\varepsilon : \mathcal{V} \to S^A$ be an interpretation of variables. Then the value $[\![t]\!]_\varepsilon$ of a relational term $t \in \mathcal{RT}(\Sigma, \mathcal{V})$ is a binary relation $(S^A)^{ar(t)} \to (S^A)^{re(t)}$ defined as follows:*

1. $[\![\mathsf{T}]\!]_\varepsilon \stackrel{\text{def}}{=} \{\Lambda\}$, $[\![\mathsf{F}]\!]_\varepsilon \stackrel{\text{def}}{=} \emptyset$;
2. $[\![I]\!]_\varepsilon \stackrel{\text{def}}{=} \boldsymbol{I}_{S^A}$;
3. *for* $f \in \Sigma$, $[\![f]\!]_\varepsilon \stackrel{\text{def}}{=} f^A$;
4. *for* $v \in \mathcal{V}$ *and a relational term* s, $[\![v.s]\!]_\varepsilon \stackrel{\text{def}}{=} \varepsilon(v)[\![s]\!]_\varepsilon$ *and* $[\![s.v]\!]_\varepsilon \stackrel{\text{def}}{=} [\![s]\!]_\varepsilon\varepsilon(v)$;
5. *for a relational term* t, $[\![\neg t]\!]_\varepsilon \stackrel{\text{def}}{=} [\![t]\!]_\varepsilon^{\mathsf{c}}$ *and* $[\![\overleftarrow{t}]\!]_\varepsilon \stackrel{\text{def}}{=} \overleftarrow{[\![t]\!]_\varepsilon}$;
6. *for relational terms* s, t, $[\![s \times t]\!]_\varepsilon \stackrel{\text{def}}{=} [\![s]\!]_\varepsilon \times [\![t]\!]_\varepsilon$, $[\![s + t]\!]_\varepsilon \stackrel{\text{def}}{=} [\![s]\!]_\varepsilon + [\![t]\!]_\varepsilon$, $[\![s \circ t]\!]_\varepsilon \stackrel{\text{def}}{=} [\![s]\!]_\varepsilon \circ [\![t]\!]_\varepsilon$, $[\![s \diamond t]\!]_\varepsilon \stackrel{\text{def}}{=} [\![s]\!]_\varepsilon \diamond [\![t]\!]_\varepsilon$, $[\![s \sqcup t]\!]_\varepsilon \stackrel{\text{def}}{=} [\![s]\!]_\varepsilon \cup [\![t]\!]_\varepsilon$ *and* $[\![s \sqcap t]\!]_\varepsilon \stackrel{\text{def}}{=} [\![s]\!]_\varepsilon \cap [\![t]\!]_\varepsilon$.

The set of connectives is not minimal — we have included almost all dual connectives to have the possibility of moving the negation "up" and "down". Only the constant I is left without its dual. Relational terms s, t are *equivalent*, written $s \equiv t$, iff, for any Σ-multialgebra A and for every interpretation of variables ε, $[\![s]\!]_\varepsilon = [\![t]\!]_\varepsilon$. We write $s = t$ if s and t (possibly variables) are syntacticaly *identical* terms.

Definition 2.4 *A Σ-multialgebra A* satisfies *a clause C if, for every interpretation of variables ε, there exists an atom $t \in C$ such that $t \equiv \mathsf{T}$.*

A satisfies *an atom a iff it satisfies the clause $\{a\}$, and* satisfies *a specification* Spec *iff it satisfies all clauses in* Spec*.*

Thus, the semantics of relational language is very similar to the semantics of nondeterministic specifications. Before comparing the two languages and translating the latter into the former, we make a brief digression.

2.3 Some equivalences

There are many equivalences relating relational terms which are well known in the theory of relations. Many of them are included in Table 1 as rewriting rules which relate equivalent terms. Here are some other laws:

Units of composition and co-composition:

$$I^{ar(t)} \circ t \equiv t \equiv t \circ I^{re(t)}, \qquad \neg I^{ar(t)} \diamond t \equiv t \equiv t \diamond \neg I^{re(t)},$$

Distributivity:

$$\begin{aligned} (s \sqcap t) \sqcup u &\equiv (s \sqcup u) \sqcap (t \sqcup u), & s \sqcup (t \sqcap u) &\equiv (s \sqcup t) \sqcap (s \sqcup u), \\ (s \sqcup t) \circ u &\equiv (s \circ u) \sqcup (t \circ u), & s \circ (t \sqcup u) &\equiv (s \circ t) \sqcup (s \circ u), \\ (s \sqcap t) \diamond u &\equiv (s \diamond u) \sqcap (t \diamond u), & s \diamond (t \sqcap u) &\equiv (s \diamond t) \sqcap (s \diamond u), \end{aligned}$$

If g is a function, then for all s :

$$g \circ s \equiv \neg g \diamond s, \qquad s \circ \overleftarrow{g} \equiv s \diamond \neg \overleftarrow{g},$$

Also, all binary connectives are associative, and $\sqcup, \sqcap$ and $\neg$ obey other laws of boolean algebras.

Many relational laws are important for "calculational" purposes [ABHVW, Mo92] but are out of scope of the current paper. We just note here, that the most convenient formulation may use the inclusion connective $\sqsubseteq$:

$$s \sqsubseteq t \equiv \begin{cases} \mathsf{T}, & \text{if } s \sqcup t \equiv t, \\ \mathsf{F}, & \text{otherwise,} \end{cases}$$

which always forms atoms. The monotonicity of connectives (all connectives but negation and inclusion are monotonic) is then expressible as

$$(s \sqsubseteq t) \sqsubseteq ((s \circ u) \sqsubseteq (t \circ u)),$$

and so on. The most important among the calculation laws are *Galois connections* [ABHVW], for example,

$$(s \circ t) \sqsubseteq u \equiv t \sqsubseteq (\neg \overleftarrow{s} \diamond u) \equiv s \sqsubseteq (u \diamond \neg \overleftarrow{t}).$$

2.4 Translation of nondeterministic specifications to relational terms

Terms, atoms and literals of nondeterministic specifications can be directly translated to relational terms of the same signature. Let $\rho(w)$ be a relational term obtained by translating word w. The translation function $\rho(\_)$ is defined inductively on the structure of w as follows:

1. for a variable $x \in \mathcal{V}$, $\rho(x) \stackrel{\text{def}}{=} x.I$;
2. for a constant $c \in \Sigma$, $\rho(c) \stackrel{\text{def}}{=} c$;
3. for a term $t = f(t_1, \ldots, t_n)$, where $ar(f) = n > 0$, $\rho(t) \stackrel{\text{def}}{=} (\rho(t_1) \times \cdots \times \rho(t_n)) \circ f$;
4. for an atom $a = s \prec t$, $\rho(a) \stackrel{\text{def}}{=} \neg\rho(s) \diamond \overleftarrow{\rho(t)}$;
5. for an atom $a = s \frown t$, $\rho(a) \stackrel{\text{def}}{=} \rho(s) \circ \overleftarrow{\rho(t)}$;
6. for an atom $a = s \approx t$, $\rho(a) \stackrel{\text{def}}{=} \neg\rho(s) \diamond I \diamond \overleftarrow{\neg\rho(t)}$;
7. for a negated atom $\neg a$, $\rho(\neg a) \stackrel{\text{def}}{=} \neg\rho(a)$;

It is interesting to note here that both intersection and functional application to arguments are translated to composition of relations. The translation of equations is a bit complicated — it demonstrates that the deterministic equality $\approx$ is some hidden form of the identity relation I.

The correctness of translation could be proved comparing semantics of the original word w and of the relational term $\rho(w)$. For that we also need translate the definition of semantics. This is rather trivial, because in the case of any term t, $[\![t]\!]_\varepsilon$ is a set, while $[\![\rho(t)]\!]_\varepsilon = \{\Lambda\} \times [\![t]\!]_\varepsilon$ is a relation of arity $\langle 0, 1\rangle$. The literals are obviously translated to relational terms of the atom kind. It is not hard to check all cases of Definition 1.2 and prove the equivalence between a literal and the translated term.

3 Complete axiomatisation of valid relational clauses

We present here the complete calculi in the style of [RS63] for establishing validity of relational clauses. We are influenced by work [BK95] of M. Białasik and B. Konikowska.

3.1 Rewriting relation

Because of multiplicity of connectives in our language, there should be quite a lot of logical rules. We try to reduce their number introducing a *rewriting relation* between relational terms. The relation is defined by *rewriting rules*, which rewrite terms to equivalent ones. Their role is quite simple: to push negation down to functional symbols, reversing sings and variable strings, and making some additional simplifications. In particular, the rules allow to eliminate reverse at all: at the moment when there is enough variables "around" a reversed functional symbol to form an atom, this atom can be reversed (the rules with α and β in Table 1). To formulate this particular kind of rules, we need to denote multiple variables surrounding a relational term, like $x.y.x.F.y.z$ for F of arity $3 \to 2$. Let us call words like $x.y.x$ *strings* of variables. Then a statement "α is a string of length n" means that α is a sequence of n variables from $\mathcal{V}$ (repetitions are allowed) separated by dot '.'. The reversed (as a word) string is denoted $\overleftarrow{\alpha}$.

Each rewrite rule has the form $s \longrightarrow t$, the set $\mathcal{R}$ of rules is given in Table 1.

Definition 3.1 *For two relational terms s, t, $s \Rightarrow t$ means that there exists a position p in s and a rule $s|_p \longrightarrow u \in \mathcal{R}$ such that $t = s[u]_p$, i.e., the term t is obtained from s by replacement of the subterm $s|_p$ by the term u. $\Rightarrow^+$ denotes the transitive closure of the relation $\Rightarrow$.*

3.2 Proof system with invertible rules

The calculus derives *valid* clauses, *i.e.*, clauses which are satisfied in any multialgebra. We consider clauses as atom sets, so repetitions do not matter. However, for the completeness proof it is convenient to fix some order of atoms in any clause. Therefore we present axioms and rules in the form which assumes such an order, for example, a clause schema C, r, D, s presents clauses with some (possibly empty) set C of atoms to the left of a term r, then there is some set D of atoms between r and s, and s is the last atom in the clause. For an unordered version, the same set of clauses as above could be presented by a schema C, r, s. We do not want to waste space for presenting both versions – an unordered variant may be easily obtained from the ordered one presented here.

We also assume that, as in Table 1, α and β denote finite (possible empty) strings of variables, with length defined by arity of the neigbouring relational term (variables are not relational terms, and variables cannot be surrounded by two terms, so there cannot be any ambiguity).

$$
\begin{array}{rlrl}
\overleftarrow{s+t} \longrightarrow \overleftarrow{t} + \overleftarrow{s}, & & \neg(s+t) \longrightarrow (\neg s) \times (\neg t), & \\
\overleftarrow{s \times t} \longrightarrow \overleftarrow{t} \times \overleftarrow{s}, & & \neg(s \times t) \longrightarrow (\neg s) + (\neg t), & \\
\overleftarrow{s \circ t} \longrightarrow \overleftarrow{t} \circ \overleftarrow{s}, & & \neg(s \circ t) \longrightarrow (\neg s) \diamond (\neg t), & \\
\overleftarrow{s \diamond t} \longrightarrow \overleftarrow{t} \diamond \overleftarrow{s}, & & \neg(s \diamond t) \longrightarrow (\neg s) \circ (\neg t), & \\
\overleftarrow{s \sqcup t} \longrightarrow \overleftarrow{s} \sqcup \overleftarrow{t}, & & \neg(s \sqcup t) \longrightarrow (\neg s) \sqcap (\neg t), & \\
\overleftarrow{s \sqcap t} \longrightarrow \overleftarrow{s} \sqcap \overleftarrow{t}, & & \neg(s \sqcap t) \longrightarrow (\neg s) \sqcup (\neg t), & \\
x.(s+p) \longrightarrow s + (x.p), & & x.(s+r) \longrightarrow (x.s) + r, & \\
(o+t).y \longrightarrow (o.y) + t, & & (q+t).y \longrightarrow q + (t.y), & \\
x.(s \times p) \longrightarrow s \times (x.p), & & x.(s \times r) \longrightarrow (x.s) \times r, & \\
(o \times t).y \longrightarrow (o.y) \times t, & & (q \times t).y \longrightarrow q \times (t.y), & \\
x.(s \circ t) \longrightarrow (x.s) \circ t, & & (s \circ t).y \longrightarrow s \circ (t.y), & \\
x.(s \diamond t) \longrightarrow (x.s) \diamond t, & & (s \diamond t).y \longrightarrow s \diamond (t.y), & \\
x.(s \sqcup t) \longrightarrow (x.s) \sqcup (x.t), & & (s \sqcup t).y \longrightarrow (s.y) \sqcup (t.y), & \\
x.(s \sqcap t) \longrightarrow (x.s) \sqcap (x.t), & & (s \sqcap t).y \longrightarrow (s.y) \sqcap (t.y), & \\
\alpha.(\overleftarrow{f}).\beta \longrightarrow \overleftarrow{\beta}.f.\overleftarrow{\alpha}, & & \alpha.(\neg\overleftarrow{f}).\beta \longrightarrow \overleftarrow{\beta}.(\neg f).\overleftarrow{\alpha}, & \\
x.(t.y) \longrightarrow x.t.y, & & (x.t).y \longrightarrow x.t.y, & \\
\overleftarrow{t.y} \longrightarrow y.\overleftarrow{t}, & \neg(t.y) \longrightarrow (\neg t).y, & x.r \longrightarrow r, & q.y \longrightarrow q, \\
\overleftarrow{x.t} \longrightarrow \overleftarrow{t}.x, & \neg x.t \longrightarrow x.(\neg t), & \overleftarrow{I} \longrightarrow I, & \neg\overleftarrow{I} \longrightarrow \neg I, \\
x.I.x \longrightarrow \mathsf{T}, & \overleftarrow{(\neg t)} \longrightarrow \neg(\overleftarrow{t}), & \overleftarrow{\overleftarrow{t}} \longrightarrow t, & \neg(\neg t) \longrightarrow t, \\
\overleftarrow{\mathsf{T}} \longrightarrow \mathsf{T}, & \neg\mathsf{T} \longrightarrow \mathsf{F}, & \overleftarrow{\mathsf{F}} \longrightarrow \mathsf{F}, & \neg\mathsf{F} \longrightarrow \mathsf{T},
\end{array}
$$

where $x, y \in \mathcal{V}$, $o, p, q, r, s, t \in \mathcal{RT}(\Sigma, \mathcal{V})$, $re(o) > 0$, $ar(p) > 0$, $re(q) = 0$, $ar(r) = 0$, $f \in \Sigma$, α and β are strings of variables of length $re(f)$ and $ar(f)$, respectively.

Table 1: The set of rewriting rules $\mathcal{R}$

3.2.1 Axioms

The following two schemata define the axiomatic clauses:

$$C,\ \mathsf{T},\ D \quad \text{and} \quad C,\ \alpha.f.\beta,\ D,\ \alpha.(\neg f).\beta,\ E, \tag{1}$$

where $f \in \Sigma$. Pairs of atoms of kind $\alpha.f.\beta$ and $\alpha.(\neg f).\beta$ are called *contrary pairs*. So, for a clause to be an axiom means to contain the atom T or some contrary pair. It is obvious, that axioms are valid clauses.

3.2.2 Proof rules

The following rule (schema) incorporates rewriting rules from $\mathcal{R}$ into our proof system:

$$\frac{C,\ t,\ D}{C,\ s,\ D}, \tag{2}$$

for any relational terms s, t such that $s \Rightarrow^+ t$

$$\frac{C,\ s.\beta,\ D,\ s \circ t \qquad C,\ \beta.t,\ D,\ s \circ t}{C,\ s \circ t,\ D}, \qquad \frac{C,\ s.\alpha,\ \alpha.t,\ D}{C,\ s \diamond t,\ D}, \tag{3}$$

$$\frac{C,\ s,\ D \qquad C,\ t,\ D}{C,\ s \sqcap t,\ D}, \qquad \frac{C,\ s,\ t,\ D}{C,\ s \sqcup t,\ D}, \tag{4}$$

$$\frac{C,\ s,\ D \qquad C,\ t,\ D}{C,\ s \times t,\ D}, \qquad \frac{C,\ s,\ t,\ D}{C,\ s + t,\ D}, \tag{5}$$

$$\frac{C,\ D}{C,\ \mathsf{F},\ D}, \qquad \frac{[C,\ D]^x_y}{C,\ x.(\neg I).y,\ D}, \tag{6}$$

where α in (3) consists of all distinct variables that do not occur in the conclusion of the rule, and $[C,\ D]^x_y$ in (6) denotes the clause obtained from $C,\ D$ by substitution of x for all occurences of y. This variable replacement is also defined on the set of relational terms with the set of rewriting rules presented in Table 2. For a clause C, $[C]^x_y \stackrel{\text{def}}{=} \{[t]^x_y : t \in C\}$. The rules should be also included into the set $\mathcal{R}$.

$$\begin{array}{rlrl}
[t]^x_x & \longrightarrow t, & & \\
[I]^y_x & \longrightarrow I, & [f]^y_x & \longrightarrow f, \\
[x.t]^y_x & \longrightarrow y.[t]^y_x, & [t.x]^y_x & \longrightarrow [t]^y_x.y, \\
[z.t]^y_x & \longrightarrow z.[t]^y_x, & [t.z]^y_x & \longrightarrow [t]^y_x.z, \\
[\neg t]^y_x & \longrightarrow \neg[t]^y_x, & [\overleftarrow{t}]^y_x & \longrightarrow \overleftarrow{[t]^y_x}, \\
[s \diamondsuit t]^y_x & \longrightarrow [s]^y_x \diamondsuit [t]^y_x, & &
\end{array}$$

where $x, y, z \in \mathcal{V}$, $z \neq x$, $f \in \Sigma$, and $\diamondsuit \in \{\circ, \diamond, +, \times, \sqcup, \sqcap\}$.

Table 2: Rules for computation of variable replacement

As in Gentzen-style sequential calculus, so in the above system, the usual way to obtain proof of validity of some clause C is to start from C and to build a proof bottom-up. The essential property of presented rules is that they are *invertible* in the sense of Lemma 3.2, what means that there can be no "wrong" steps in a proof which would make the proof impossible. Of course, some proof steps may be useless.

Lemma 3.2 *For all rules (2)–(6), the conclusion is valid iff all the premises are valid.*

Proof. Both sides of rewriting rules are equivalent terms, what means that lemma is satisfied in the case of rule (2). Four rules form (4) and (5) correspond to analogous rules for logical conjuction and disjunction, because in the case of atoms $\sqcup$ and $+$ behave like disjunction, while $\sqcap$ and $\times$ like conjunction. The rule with F is trivial and, in fact, is not needed for completeness.

The rule with $\neg I$ follows from the observation, that $x.I.y$ iff $x = y$. The rules from (3) are most complicated, because they subsume quantifier rules of the first-order logic. Therefore, the rule with $\diamond$ requires new variables in the premise, like the analogous rule for universal quantification. The rule for $\circ$ requires to repeat in the premise the term from the conclusion, because this is the only rule, in which we must guess values meant by existential quantification, which is hidden in composition. $\square$

The notion of proof is defined as usual. The completeness result is as follows:

Theorem 3.3 *A clause is valid iff it has a proof in the presented proof system.*

The proof is presented in Appendix A

4 Adding λ-abstraction

The introduced language of relational terms have functional symbols with arguments firmly divided into two parts. This is not always convenient when we are trying to specify applications described with usual relations, without partition of the arguments. Then, in formulae, we need different partitions of the same relations. It is possible to introduce many duplicates of the same relation, one for each necessary partition of arguments, but this is a hard way. Another problem is the fixed order of arguments: sometimes we need to compose relations with some changed argument order, and this also causes the similar problems as above.

One way to solve this problem would be to use some graphical interface language: it is easy to present any connection of arguments in the two dimensional plane. Then, some graph language should be used for formalisation. But we would like to have a more traditional "string-oriented" language. We found a convenient solution of the argument manipulation problems in introduction of the λ-abstraction construction, similar to the functional λ-calculus. We use the same notation and name of the construction, because there is a close correspondence (at least at the syntactic level) with λ-calculus. One should however bear in mind that we consider more general objects than functions — relations — but with the restriction that values of variables are only carrier elements. Without this last restriction we would have generalization of λ calculus to the case of relations (or nondeterministic functions).

The syntax of relational terms is extended with the following construction:

Definition 4.1 *For any relational term t and any variable x, $\lambda x.t$ and $t.x\lambda$ are relational terms of arities $ar(t) + 1 \rightarrow re(t)$ and, respectively, $ar(t) \rightarrow re(t) + 1$.*

The semantics of this construction is defined (in terms of Definition 2.3) as follows:

Definition 4.2 $[\![\lambda x.t]\!]_\varepsilon \stackrel{\text{def}}{=} \{\langle ua, v\rangle : \langle u, v\rangle \in [\![t]\!]_{\varepsilon\{x\mapsto a\}}\}$, *where* $\varepsilon\{x \mapsto a\}$ *is just* ε *only updated to map* x *to a value* a*; similarly,* $[\![t.x\lambda]\!]_\varepsilon \stackrel{\text{def}}{=} \{\langle u, av\rangle : \langle u, v\rangle \in [\![t]\!]_{\varepsilon\{x\mapsto a\}}\}$.

For λ-abstraction $\lambda x.t$ or $t.x\lambda$, there is no syntactic restriction of the kind "x must be a free variable in t", and a fictive argument x may be introduced in this way, for example, the relation U defined by

$$U \stackrel{\text{def}}{=} \mathsf{T}.x\lambda \tag{7}$$

is of type $0 \to 1$ and will be used below. Also, the multiple occurences of the variable x in both sides of the relational term t may be resumed in one argument. Sequences of abstractions can rearrange argument order, and even change sides where arguments occur. These tricks are used in the following translation of the language of first-order predicate calculus to the language of relational terms.

4.1 Translation of first-order predicate language

Let us fix some signature of first-order predicate language, consisting of finite sets $\mathcal{F}$ and $\mathcal{P}$ of functional and predicate symbols, and let $\Sigma = \mathcal{F} \cup \mathcal{P}$ be the signature of relational terms to which formulae and terms of predicate language are translated. We assume that predicate formulae are constructed using only logical connectives $\wedge$, $\vee$ and $\neg$. In order to obtain shorter relational terms after translation, we also assume that quantified first-order formulae have some restricted form (or have been transformed to this form). Indeed, if $\forall x A$ is a formula, then A contains free occurences of x and is of one of the following forms:

- $\forall y B$ or $\exists y B$ for some B (satisfying similar requirements);
- $B \vee C$, where both B and C contain free occurences of x;
- A does not contains binary connectives.

In the case of a formula $\exists x A$, the second form is replaced by $B \wedge C$, while the rest remains the same. These restrictions collect all the cases when a quantifier can not be "moved further down" without changing the meaning of the whole formula. For example, $\forall x(B \wedge C)$ is equivalent to $\forall x B \wedge \forall x C$, where quantifiers occur at one level lower than before, but an analogous transition when $\wedge$ is replaced by $\vee$ does not ensure equivalence.

We use the same translation function $\rho(.)$ as in Section 2.4 where nondeterministic terms and formulae were translated, because terms are translated in the same way. The translation function is defined (ambigiously) inductively on the structure of formulae as follows:

1. $\rho(s = t)$ is either $s.I.t$ if both s, t are variables, or $(s.I) \circ \rho(t)$, if s is a variable and t is not, or $\rho(s) \circ I \circ \rho(t)$ if both s, t are not variables;
2. $\rho(\forall x(F \vee G)) \stackrel{\text{def}}{=} ((\rho(F)).x\lambda) \diamond (\lambda x.(\rho(G)))$;
   $\rho(\forall x F) \stackrel{\text{def}}{=} U \diamond (\lambda x.(\rho(F)))$;
3. $\rho(\exists x(F \wedge G)) \stackrel{\text{def}}{=} ((\rho(F)).x\lambda) \circ (\lambda x.(\rho(G)))$;
   $\rho(\exists x F) \stackrel{\text{def}}{=} U \circ (\lambda x.(\rho(F)))$;
4. $\rho(F \vee G) \stackrel{\text{def}}{=} \rho(F) \sqcup \rho(G)$;
   $\rho(F \wedge G) \stackrel{\text{def}}{=} \rho(F) \sqcap \rho(G)$;
5. $\rho(\neg F) \stackrel{\text{def}}{=} \neg\rho(F)$,

where U was defined by (7).

For the proof of correctness of translation, a translation of semantics would be needed, too. We suppose that writing down the definition of semantic translation and of the related proof is quite obvious and we do not present them here.

4.2 Supplement to the rewriting rules

The proof system presented above remains complete after the extension of the language with λ-abstraction, if the rewriting rule system $\mathcal{R}$ is extended with the rules presented in Table 3.

$$
\begin{array}{rclrcl}
\neg(\lambda x.t) & \longrightarrow & \lambda x.(\neg t), & \neg(t.x\lambda) & \longrightarrow & (\neg t).x\lambda, \\
\overleftarrow{\lambda x.t} & \longrightarrow & \overleftarrow{t}.x\lambda, & \overleftarrow{t.x\lambda} & \longrightarrow & \lambda x.\overleftarrow{t}, \\
y.(\lambda x.t) & \longrightarrow & [t]^y_x, & (t.x\lambda).y & \longrightarrow & [t]^y_x, \\
z.(t.x\lambda) & \longrightarrow & (z.t).x\lambda, & (\lambda x.t).z & \longrightarrow & \lambda x.(t.z), \\
x.(t.x\lambda) & \longrightarrow & (x.[t]^v_x).v\lambda, & (\lambda x.t).x & \longrightarrow & \lambda v.([t]^v_x.x),
\end{array}
$$

where z and v are variables different from x and v does not occur in t.

Table 3: Rewriting rules for terms with λ

There are β-reduction rules between them, they are applied to variables only because of our restrictions for variables to have only elements as values. Deterministic terms could be substituted in variables, but this kind of modification is out of scope of this paper.

References

[ABHVW] C.J. Aarts, R.C. Backhouse, P. Hoogendijk, T.S. Voermans, and J. van der Woude. A Relational Theory of Datatypes. September 1992. Available via World-wide web at http://www.win.tue.nl/win/cs/wp/papers.

[BK95] M. Białasik, B. Konikowska. A Logic for Nondeterministic Specifications. Submitted to *Journal of Nonclassical and Applied Logic*.

[Hes88] W.H. Hesselink. A Mathematical Approach to Nondeterminism in Data Types. *ACM Transactions on Programming Languages and Systems*, 10, pp.87-117, (1988).

[Hus92] H. Hussmann. Nondeterministic algebraic specifications and nonconfluent term rewriting. *Journal of Logic Programming*, 12, pp.237-235, (1992).

[Hus93] H. Hussmann. *Nondeterminism in Algebraic Specifications and Algebraic Programs.* Birkhäuser Boston, (1993).

[Kap88] S. Kaplan. Rewriting with a Nondeterministic Choice Operator. *Theoretical Computer Science*, 56:1, pp.37-57, (1988).

[KW94] V. Kriaučiukas, M. Walicki. Reasoning and Rewriting with Set-Relations I: Ground-Completeness. In *Proceedings of CSL'94*, LNCS 933, pp. 264–278 (1995). Also: Report no.96, Dept. of Informatics, University of Bergen (1994).

[KW95] V. Kriaučiukas, M. Walicki. Reasoning and Rewriting with Set-Relations II: The Non-Ground Completeness. In *Proceedings of ADT'11, Soria-Moria, Oslo, September 19–24, 1995.* LNCS, Springer-Verlag (to appear)

[Mo92] O. de Moor. *Categories, Relations and Dynamic Programming.* D.Phil Thesis. PRG Technical Monograph PRG-98, Programming Research Group, Oxford University Computing Laboratory (1992).

[Pr88] V. Pratt. Dynamic algebras as a well-behaved fragment of relation algebras. In C.H. Bergman, R.D. Maddux, D.L. Pigozzi (Eds.). *Algebraic Logic and Universal Algebra in Computer Science*, Proc. of Conf., Ames, Iowa, USA, June 1–4, 1988. *Lect.Notes in Comp.Sci* Springer-Verlag, 425, pp.77–110, (1990).

[RS63] H. Rasiova, R. Sikorski. *The Mathematics of Metamathematics.* Polish Scientific Publishers, Warsaw (1963).

[WM95a] M. Walicki, S. Meldal. Multialgebras, Power algebras and Complete Calculi of Identities and Inclusions. In *Recent Trends in Data Type Specification, Selected papers of joint 10th ADT and 5th COMPASS workshops, S. Margherita, Italy, May 30 - June 3, 1994*, Springer-Verlag, LNCS 906, pp. 453–468 (1995).

[WM95b] M. Walicki, S. Meldal. A Complete Calculus for Multialgebraic and Functional Semantics of Nondeterminism. *ACM Trans. on Programming Languages and Systems*, vol. 17, N0. 2, pp. 366–393 (1995).

A The proof of completeness

Lemma 3.2 implies soundness of the proof rules, *i.e.*, if there exists a proof of a clause C, then the clause C is valid. As usual, the hardest part is completeness: to show that if there does not exist a proof of C then there does exist a model (multialgebra) not satisfying C. And what is quite usual for this kind of proofs of model existence, is to solve an even harder problem: to show, that if C does not have a proof built according to some given strategy, then there exists a model not satisfying C.

A.1 Proof strategy

The strategy used in our case supposes some order of atoms in any clause. According to this strategy, inference rules are always applied (bottom-up) to the first atom (from the left, or in some other ordering) that allows application of some rule. The only atoms to which rules can not be applied are

$$x.I.y, \qquad \alpha.f.\beta, \qquad \alpha.(\neg f).\beta, \tag{8}$$

where $f \in \Sigma$, α and β are variable strings of length, respectively, $ar(f)$ and $re(f)$.

All rules except two – the rewriting rule (2), and the $\circ$-rule (3) – are *deterministic*, in the sense, that their conclusion defines premisses unambigiously, in the case of $\diamond$-rule, up to renaming of new variables.

The rule (2) involving rewriting can be made deterministic too, because of the following result.

Lemma A.1 *The rule system $\mathcal{R}$ is terminating and confluent, i.e., for any relational term t there is no infinite sequence $t \Rightarrow t_1 \Rightarrow \cdots$ of rewritings, and, if t can be rewritten in $\mathcal{R}$, then there exists a unique term t' such that $t \Rightarrow^+ t'$ and t' can not be rewritten in $\mathcal{R}$.*

Proof. Proof by induction on the levels of occurrences of negation, reverse and variables in t. $\square$

We can add to our strategy the requirement, that in the proof rule with rewritings, t should be in *normal form*, *i.e.*, can not be rewritten in $\mathcal{R}$. Then uniqueness of the normal form guarantees determinism of this proof rule.

Only the $\circ$-rule is a possible source of infinite proofs. The strategy must be *fair* with respect to this rule, i.e., if there is an infinite branch in some proof-tree built according to the strategy, and $\mathcal{X}$ is the set of all variables occurring in clauses along this branch, then for any term $s \circ t$ that occurs in the branch and for any string γ of length $re(s)$ with variables from $\mathcal{X}$, there exists in the branch an application of $\circ$-rule to $s \circ t$ with $\beta = \gamma$. We will construct the carriers of the model from variables, and we must be sure that the $\circ$-rule along an infinite branch exhausts all possible combinations of carrier elements. This exhaustive search can be fixed in the strategy, because $\mathcal{X}$ is countable and hence $\mathcal{X}^{re(s)}$ is countable, too. In this way the $\circ$-rule also can be made deterministic. This determinization assumes some variable ordering, also needed for determinisation of the $\diamond$-rule.

As a result, we may assume, that our strategy is deterministic and fair.

A.2 Premultialgebras

Premultialgebras, which are derivable from sets of atoms of the forms listed in (8), will help us in the construction of countermodel.

Definition A.2 *A* Σ-premultialgebra *is a triple* $P = \langle S^P, \Sigma^P, I_-^P \rangle$, *where* S^P *is a* carrier, Σ^P *is a set of binary relations* $\{f_-^P, f_+^P : (S^P)^{ar(f)} \to (S^P)^{re(f)} : f \in \Sigma\}$ *such that* $f_-^P \cap f_+^P = \emptyset$ *and* $I_-^P \subseteq S^P \times S^P$ *is a binary relation,* $x.I_-^P.y \Rightarrow x \neq y$.

For a given set C of atoms each having a form listed in (8), and not containing a contrary pair, a premultialgebra $\mathsf{Pre}(C)$ is constructed in the following way: the carrier consists of all the free variables occuring in C, I_- is defined by $xy \in I_- \iff x.I.y \in C$, and the rest of relations is defined as follows (keeping in mind 1-1 correspondence between variable strings and sequences and therefore allowing ourselves some unstrictness in notations)

$$\alpha.\beta \in f_- \iff \alpha.f.\beta \in C, \tag{9}$$

$$\alpha.\beta \in f_+ \iff \alpha.(\neg f).\beta \in C. \tag{10}$$

If C does not contain a contrary pair, then $\mathsf{Pre}(C)$ is a premultialgebra.

When in a Σ-premultialgebra P, for all $f \in \Sigma : (f_+^P)^{\mathsf{c}} = f_-^P$, and $I^{\mathsf{c}} = I_-^P$, then P determines a Σ-multialgebra. A multialgebra can be obtained by choosing the interpretation of f as either f_+^P or as f_-^P. Choosing the former interpretation (of f as f_+^P) will yield a multialgebra which does not satisfy C, and it is this interpretation we will have in mind in the sequel.

The following result is quite obvious, and we present it without proof.

Lemma A.3 *Each Σ-premultialgebra can be completed to a Σ-multialgebra. More specifically, given a clause C of atoms having the forms from (8) which does not contain a contrary pair,* $\mathsf{Pre}(C)$ *can be completed to a multialgebra which does not satisfy C.*

For a function $h : S \to T$ and a sequence $a_1 \ldots a_k \in S^*$ let us write $h(a_1 \ldots a_k)$ for the sequence of images $h(a_1) \ldots h(a_k)$, and for a subset $S' \subseteq S$, let $h(S') \stackrel{\text{def}}{=} \{h(a) : a \in S'\}$.

Definition A.4 *For two Σ-premultialgebras P and Q, a function $h : S^P \to S^Q$ is a* pre-homomorphism, *denoted $h : P \to Q$, iff $h(I_-^P) \subseteq I_-^Q$ and, for all $f \in \Sigma$, $h(f_+^P) \subseteq f_+^Q$ and $h(f_-^P) \subseteq f_-^Q$.*

For example, an embedding of a premultialgebra P into P' obtained from P according to Lemma A.3 is a pre-homorphism. Pre-homomorphisms are closed under composition, what follows directly from Definition A.4. For us, the following result is essential:

Lemma A.5 *For any (finite or infinite) sequence of Σ-premultialgebras connected sequentially by pre-homomorphisms:*

$$P_0 \xrightarrow{h_0} P_1 \xrightarrow{h_1} P_2 \xrightarrow{h_2} \cdots$$

there exists a Σ-premultialgebra P and pre-homomorphisms $g_i : P_i \to P$ such that $g_i = h_{i+1}; g_{i+1}$.[1]

Proof. The construction is quite standard. Form the set S as a disjoint union of all carriers S^{P_i}, and let the relation H be the union of all functions h_i considered as relations. Let $E \subseteq S \times S$ be the reflexive, symmetric and transitive closure of H, and S^P be the set of all equivalence classes over S wrt. E. Then define f_-^P, f_+^P and I_-^P on these classes, if the corresponding relation holds for some representatives in some of premultialgebras P_i. Monotonicity along the whole sequence ensured by the existing pre-homomorphisms implies that requirements for P to be a premultialgebra are satisfied. (In particular, $f_-^P \cap f_+^P = \emptyset$, for if not, then these two sets would intersect already for some P_i. A similar compactness argument shows that I_-^P does not relate equal elements.)

The required pre-homomorphisms g_i simply map elements to their classes.

$\square$

[1] The order of composition is diagramatic.

A.3 Model from variables

Assume that a clause C does not have a proof. Then a proof-tree built according to the strategy is not a proof, i.e., it contains a branch which either is infinite or else ends with a clause that is not an axiom and consists only of atoms of the forms listed in (8). In the latter case the clause D ending the branch does not contain a contrary pair, so $\mathsf{Pre}(D)$ can be completed to a multialgebra, which does not satisfy D by Lemma A.3 and, by Lemma 3.2, does not satisfy C.

In the case of an infinite branch, let $C = C_0, C_1, \ldots, C_k \ldots$ be the sequence of the clauses in this branch, $D_i \subseteq C_i$ be the set of all atoms from C_i that have the forms listed in (8). Each D_i does not contain a contrary pair and therefore $\mathsf{Pre}(D_i)$ is a premultialgebra. Variable *embedings* are pre-homomorphisms in cases of all rules except the rule for $\neg I$, because this is the only rule which can change D_i renaming its variables. This rule replaces one of the variables from the active atom $x.(\neg I).y$ by the other one. We therefore have to map both variables x, y from this atom to the one which is left, and all the remaining variables onto themselves. It is easy to see that such a map will be a pre-homomorphism.

This means that the sequence of premultialgebras $\mathsf{Pre}(D_0), \mathsf{Pre}(D_1), \ldots$ is connected by pre-homomorphisms. Lemma A.5 states, for such a case, the existence of a premultialgebra with a pre-homomorphism from each $\mathsf{Pre}(D)_i$, which therefore does not satisfy any of D_i. By Lemma A.3, we can conclude the existence of a multialgebra M that does not satisfy any D_i. The pre-homomorphisms whose existence is stated in Lemma A.5 are variable interpretations giving necessary counterexamples.

To show that M also does not satisfy any C_i, induction on the length of formulae is used. Special treatment is required for the formulae $s \circ t$, but their unsatisfiability follows easily from the fairness of the strategy.