

Relational Programming.

M.A.Malkov

Russian Research Center for Artificial Intelligence.

E-mail: ma@malkov.msk.su

Contents

1. Construction of Knowledge Bases	90
1.1. Theorem Proving	90
1.2. Equivalential Calculus	91
2. Data Base Construction	93
3. Rule of Sequential Removing of Atoms	93
References	96

The programming language is called relational, if this language has means to construct procedures and no means to construct functions. Programming with usage of relational languages is called relational.

A relational language belongs to the second stage. This means a relational language use an abstract computer without memory.

In programming, procedures are relations establishing connections between parameters of these procedures.

These connections in relations can be functional. Then part of procedure parameters is arguments of functions, the remaining arguments are values of these functions. As a rule, a procedure has functions with identical arguments. Any procedure can also contain a single function.

Functional connections can be missing in a procedure. Then the procedure is a relation selecting acceptable values of arguments.

In both cases a relation is interpreted as a table. A number of columns in this table equals a number of arguments. Each line of the table defines admissible combinations of parameter values. The table can consist of one line.

Procedure parameters have values. For calculation of these values functions can be used. These functions belong to a fixed set. In particular this set can include arithmetical operations.

Below procedure parameters are called terms, and procedures are called relations. The arbitrary terms are denoted by t_1, t_2, \dots , arbitrary relations - by $R_1(t_1, t_2, \dots), R_2(t_1, t_2, \dots), \dots$.

The usage of relations expands the field of applicability of relational programming.

In the simplest case the program consists of the sequence of independent relations. Each relation in this case is a statement.

In more complex cases we can construct sentences from relations using logic operations of disjunction, conjunction, implication, equivalence, negation, and also using universal and existential quantifiers.

Such sentences are the statements of the language. The set of these sentences forms a program.

There are very many programs solving the same problem. The set of these programs is reduced to minimum in relational programming.

To reach this purpose, programs have the next restriction: the result of execution of the program should be independent of the sequence of execution of the language sentences (but the program effectiveness essentially depends on this sequence). This restriction is realized by program presentation in the special (normal) form.

One more feature of relational programming consists of the usage of an abstract computer without memory. This means, that we abstract from the way of table presentation in computer memory and from the table distribution in a computer net.

The programs, made according to the rules of relational languages, solve two main problems - constructing of knowledge bases (theories) and data bases (models).

Each independent knowledge is interpreted as an axiom of some theory. The knowledge, deduced from other knowledge, is interpreted as a theorem of this theory. The knowledge, generalized another knowledge, is interpreted as definition.

The most popular problem of knowledge bases is theorem proving, i.e. proving that some problem is solved on the basis of available knowledge. This proving simultaneously points at the sequence of steps leading to the solution of this problem.

Below construction of knowledge bases is restricted by theorem proving. But there are also different problems, for example, construction of the set of all theorems of a given theory.

The problem of constructing of data bases is a simpler one. The initial information for constructing data bases is given by tables. From these tables we construct joints, intersections, complement, projections and direct (Cartesian) products according to the statements of the programming language.

1. Construction of Knowledge Bases.

The same knowledge can be introduced by many ways. To overcome this ambiguity we use representation in the first normal form [1].

In subsection 1.1 we suppose, that any knowledge can be formalized by its presentation as a logical formula. Then logical formulas are reduced to the first normal form.

We use knowledge bases to resolve some problems. If resolved problem is formalized, then this problem becomes a theorem. Proving the theorem we can extract the solution of the problem from this proof.

In subsection 1.2 we use relation programming to proof some theorems of the equivalential calculus as an example.

1.1. Theorem Proving.

A formula (to be more exact, a logical formula) is:

- any relation $R_1(t_1, t_2, \dots)$;
- disjunction of the formulas $\mathcal{F}_1 \vee \mathcal{F}_2$ (\mathcal{F}_1 and \mathcal{F}_2 - formulas);
- conjunction of formulas $\mathcal{F}_1 \wedge \mathcal{F}_2$;
- implication of formulas $\mathcal{F}_1 \rightarrow \mathcal{F}_2$ or $\mathcal{F}_2 \leftarrow \mathcal{F}_1$;
- negation of a formula $\neg \mathcal{F}_1$;
- equivalence of formulas $\mathcal{F}_1 \Leftrightarrow \mathcal{F}_2$;
- universal quantifier before a formula $\forall x_1 \mathcal{F}_1$, x_1 - arbitrary variable;
- existential quantifier before a formula $\exists x_1 \mathcal{F}_1$.

The formulas are reduced to the first normal form by using the next sequence:

- transformation of the axioms to the closed formulas by addition of universal quantifiers for all free variables;
- transformation to the prenex form with all quantifiers in the beginning of the formula;
- reduction to $\forall\exists$ -form, i.e. partition of one sentence on some sentences with universal quantifiers preceded to existential quantifiers;
- reduction (if there is existential quantifiers) to disjunctive normal form (DNF), at reduction the properties of equalities (for simplification DNF) are used;
- import of existential quantifiers into \wedge -clauses (\wedge -clause is a literal or a sequence of literals, joint by conjunctions, literal is relation without or with negation, \wedge -clause contains variables, bound by existential quantifiers);
- reduction (with usage of the equality properties) to a conjunctive normal form (CNF); if there are existential quantifiers then their scope is considered monolithic, not destroyed at reduction to CNF;
- separation each clause of CNF in the isolate formula (CNF is a sequence of clauses, joint by conjunctions, every clause is a sequence of one \vee -clause and some \wedge -clauses, joint by disjunctions, \vee -clause is a literal or a sequence of literals, joint by disjunction too);
- introduction of new identifications of variables: all variables, bound by universal quantifiers, are replaced by x_1, x_2, x_3, \dots , the variables, bound existential quantifiers, are replaced by a_1, a_2, a_3, \dots , and the subscript numeration starts anew in each \wedge -clause;
- all quantifiers are removed.

In a greater detail about reduction to the first normal form see [1].

To proving a theorem we need to add to a knowledge base negation of a theorem. This negation must be in the first normal form as a rule. The received set of formulas is inconsistent, and the theorem proof is a proof of inconsistency of this set.

The rules of establishing inconsistency of a set of formulas are presented in [1].

The formula set, including knowledge base and negation of the theorem, is a program ready for execution. The program is executed by interpretation. The result of execution of this program is a proof of the theorem. If the theorem is some problem (more precisely, the existence of the solution of the problem), then we can extract the solution of this problem from this proving.

1.2. Equivalential Calculus.

The equivalential calculus [2] has axioms expressing properties of logical equivalence in the multi-valued logic.

The signature of the calculus includes one sort x , one symbol of two-place function e and one predicate symbol P .

The sort x is interpreted as a set of logical values. In particular it can be a set of two values "true" and "false" or a countless set.

The function e is interpreted as the equivalence relation. As logic is multi-valued, the equivalence relation is multi-valued too and transforms to the function $e : x \times x \rightarrow x$. If the sort x is finite, the function e is a truth table for the equivalence relation in the multi-valued logic.

The predicate P defines a true value of argument. I.e. members of some (nonempty) subset x are interpreted as true.

The equivalential calculus contains three axioms:

1. $P(e(x_1, x_1))$.
2. $P(e(e(x_1, x_2), e(x_2, x_1)))$.
3. $P(e(e(x_1, x_2), e(e(x_2, x_3), e(x_1, x_3))))$.

and one inference rule:

4. $\neg P(e(x_1, x_2)) \vee P(x_1) \vee P(x_2)$.

Axioms are interpreted accordingly as reflexivity, symmetry and transitivity of logical equivalence. The inference rule is interpreted as x_2 is true, if x_1 is logically equivalent to x_2 and if x_1 is true.

A number of axioms can be reduced. In particular, axiom 1 is superfluous. Moreover, a number of axioms can be reduced to one [3]. In paper [4] 5 theorems of equivalential calculus are given, some theorems consist of several parts, and each part is a theorem too.

The application of the rules of relational programming allows to find all shortest proofs of these theorems. The results of this application are shown in the next table:

Theorem	A number of shortest proofs	A step number of shortest proof	A step number of proof in [4]
1	1	2	2
2	2	6	6
3 part 1	2	3	3
3 part 2	10	8	19
4 part 1	8	8	13
4 part 2	1	7	8
4 part 3	1	7	13
5	1	5	18

As follows from this table, a number of the proof steps in theorem 3 part 2 is reduced by 2.5 times, in theorem 5 - by 3.5 times.

Let's produce proofs of these two theorems.

Theorem 5. From symmetry and associativity there follows a commuted form of associativity:

$$P(e(e(e(x_1, x_2), e(x_3, x_1)), e(x_2, x_3)))).$$

The program:

0. $\neg P(e(e(e(a_1, a_2), e(a_3, a_1)), e(a_2, a_3)))$ (negation of the theorem)
1. $P(e(e(x_1, x_2), e(x_2, x_1)))$ (symmetry)
2. $P(e(e(x_1, x_2), e(e(x_2, x_3), e(x_1, x_3))))$ (associativity)
3. $\neg P(e(x_1, x_2)) \vee \neg P(x_1) \vee P(x_2)$ (inference rule)

The proof being the result of this program execution, is reduced below. In this proof the numbers of three sentences are presented in brackets. These sentences take part in the resolution. They are: the inference rule, the sentence taking part in the resolution with the first literal of the inference rule, and the sentence taking part in the resolution with the second literal of this rule. We can show the proof of any theorem of equivalential calculus is possible to be presented in such form. So:

4. (3,1,2) $P(e(e(x_1, e(x_2, x_3))), e(e(x_1, x_2), x_3))$).
5. (3,2,2) $P(e(e(x_1, x_2), e(x_3, e(x_1, e(x_2, x_3)))))$).
6. (3,5,4) $P(e(x_1, e(e(x_2, e(x_3, x_4))), e(e(e(x_2, x_3), x_4), x_1))))$).
7. (3,4,6) $P(e(e(x_1, e(x_2, e(x_3, x_4))), e(e(e(x_2, x_3), x_4), x_1))))$).
8. (3,7,5) $P(e(e(e(x_1, x_2), e(x_3, x_1)), e(x_2, x_3)))$).

The proof is complete. We have got the source theorem.

Theorem 3 part 2. The eighth shortest axiom of equivalential calculus follows from symmetry and transitivity (there are 13 such axioms, each axiom is single in equivalential calculus, see [3]):

$$P(e(e(x_1, x_2), e(x_3, e(e(x_3, x_2), x_1))))$$

The program:

0. $\neg P(e(e(a_1, a_2), e(a_3, e(e(a_3, a_2), a_1))))$ (negation of the theorem)
1. $P(e(e(x_1, x_2), e(x_2, x_1)))$ (symmetry)
2. $P(e(e(x_1, x_2), e(e(x_2, x_3), e(x_1, x_3))))$ (transitivity)
3. $\neg P(e(x_1, x_2) \vee \neg P(x_1) \vee P(x_2))$ (inference rule)

All proofs resulting from the execution of the program, are introduced in the table:

Step	Proof									
	1	2	3	4	5	6	7	8	9	10
4	3,2,2	3,2,1	3,2,2	3,2,2	3,2,2	3,2,2	3,2,2	3,2,2	3,2,2	3,2,2
5	3,4,4	3,1,2	3,4,4	3,4,4	3,2,4	3,2,4	3,2,4	3,2,4	3,4,4	3,4,4
6	3,1,5	3,5,5	3,5,1	3,5,1	3,5,1	3,5,1	3,5,1	3,5,1	3,5,1	3,5,1
7	3,5,4	3,2,6	3,2,6	3,2,6	3,4,6	3,5,5	3,6,6	3,2,5	3,5,6	3,5,4
8	3,5,1	3,7,2	3,1,5	3,6,7	3,5,7	3,7,6	3,6,7	3,6,6	3,4,7	3,7,6
9	3,7,8	3,2,8	3,7,5	3,8,5	3,1,8	3,1,8	3,5,8	3,7,8	3,8,6	3,6,8
10	3,9,9	3,7,9	3,8,9	3,9,5	3,6,9	3,6,9	3,1,9	3,6,9	3,9,8	3,9,5
11	3,6,10	3,4,10	3,7,10	3,7,10	3,10,9	3,10,9	3,8,10	3,10,9	3,10,2	3,10,8

The next theorem has a special interest [3].

Theorem. The shortest single axiom 10 is deduced from the shortest single axiom 11.

This theorem was proved in 1978 [5] and the proof had 43 steps. In 1984 this theorem was proved with 23 steps [6]. And in 1989 this theorem was proved with 10 steps [7].

Using relation programming one can show this theorem has 5 proofs with 10 steps and no shorter proof is possible.

The program for this theorem is next.

0. $\neg P(e(e(e(a_1, e(a_2, a_3)), a_3), e(a_2, a_1))))$ (negation of the axiom 10).
1. $P(e(x_1, e(e(x_2, e(x_3, x_1))), e(x_3, x_2))))$ (axiom 11).
2. $\neg P(e(x_1, x_2) \vee \neg P(x_1) \vee P(x_2))$ (inference rule).

The result of this program execution is placed in the next table.

Step	Proof				
	1	2	3	4	5
3	2,1,1	2,1,1	2,1,1	2,1,1	2,1,1
4	2,3,1	2,3,1	2,3,1	2,3,1	2,3,1
5	2,1,3	2,1,4	2,4,4	2,3,3	2,4,3
6	2,3,5	2,3,5	2,1,5	2,1,5	2,1,5
7	2,1,6	2,1,6	2,1,3	2,6,1	2,6,1
8	2,7,1	2,7,1	2,7,1	2,3,7	2,1,7
9	2,3,8	2,3,8	2,6,8	2,3,8	2,8,1
10	2,3,9	2,3,9	2,3,9	2,4,9	2,3,4
11	2,4,10	2,4,10	2,4,10	2,1,10	2,3,10
12	2,3,11	2,3,11	2,6,11	2,11,10	2,9,11

2. Data Base Construction.

The same data base can be constructed by different ways.

In relational programming the data base should be relational too [8]. This condition essentially reduces a number of ways of presentation of data bases. More that we just abstract from the way of table presentation in computer memory and from the table distribution in a computer net.

So the data base is a set of relational tables, every table consists of a fixed number of columns and unrestricted number of lines. A data item is information stored at intersection of a column and a line. The data item can have arbitrary length.

The basic problem of data bases is construction of tables fitting to some conditions. Such tables can contain one column and one line in a simple case or be a set in explicit form, without any conditions.

As a rule, the tables are stored in data base in explicit form. The other tables can be calculated if it is necessary. If such table takes much computing time to be calculated and is used very often then this table is stored in the data base too.

But the boundaries between the explicit and calculated tables are absent in relational programming. A user gives only conditions put on the tables in a data base to extract a needed information. A user does not know these tables are stored in the data base or must be calculated. The conditions, put on the tables of the data base, are logical formulas (see above).

As it was pointed, there are many ways to present the same formula. In relational programming a logical formula should be presented in the second normal form [1].

This means the logical formula should be present at first as the definition of a relation, corresponding to needed table:

defined relation \Leftrightarrow defining formula

At second we use a positive part of this definition (positive definition):

defined relation \leftarrow the defining formula

Then we use the following rules of reduction to the second normal form [1]:

- reduction of positive definition to the first normal form with defined relation in the beginning of each formula;

- replacement of \wedge -clauses in the sentences by new relations and addition of positive definitions for these relations (\wedge -clause is literals, joint by conjunctions, literal is relation without or with negation);

- transformation of all positive definitions to back implication with the defined relation as left argument and with negation of a remaining part of the definition as right argument of back implication.

On it the construction of the second normal form ends. The right argument of back implication is a \wedge -clause and all its variables, absent in the left argument, are connected by implicit existential quantifiers.

So the second normal form is:

$$L_0 \leftarrow L_1 \wedge L_2 \wedge L_3 \dots$$

where L_i is $R_i(t_{i_1}, t_{i_2}, \dots)$ or $\neg R_i(t_{i_1}, t_{i_2}, \dots)$, R_i is a relation, t_{i_j} is j -th argument of the relation.

3. Rule of Sequential Removing of Atoms.

The second normal form can be used both to construct a data base and to formalize logical rules.

We demonstrate this as example by rule of sequential removing of atoms.

This rule is the most effective method of theorem proving in propositional logic.

The theorem proving, considered in the previous section, belongs to the first order logic.

Propositional logic is simpler than the first order logic. In particular there is no knowledge base. Sets of propositions and proofs are rather a data base. Therefore rules of theorem proving can be interpreted as constructing the tables containing these proofs. In programs these tables are presented as relations.

In propositional logic theories consist of axioms, and axioms consist of literals, joint by disjunctions [9]. A literal is an atom (elementary preposition) without negation (positive literal) or with it (negative literal). An atom in the positive literal has the sign "+", in the negative literal - the sign "-".

The axioms, atoms and their signs are encoded by natural numbers, starting from zero. This means, that axioms and elementary propositions are ordered and that a theory has a maximal atom

and maximal axiom. A number of atoms is by 1 greater than the code of a maximal atom, and a number of the axioms is by 1 greater than the code of the maximal axiom.

Negation of proving theorem is added to the set of the axioms of the theory. As a result the theory becomes inconsistent.

The inference rule is the rule of the resolution. The resolution of two axioms is a resolvent. The resolvent contains all atoms (with signs) of both axioms, except the atom of a contrary pair.

The contrary pair is an atom presented at both axioms, but with different signs. The contrary pair should be single.

The rule of sequential removing of atoms makes:

- the resolution of every pair of the axioms with a maximal atom as an atom of contrary pair;
- addition of received resolvents to the set of the axioms;
- removing from received set of the axioms of all with maximal atom or subsumed by received resolvents;
- repetition of the previous steps till a maximal atom exists or empty resolution is got.

The empty resolvent is received at the resolution of the monatomic axioms, i.e. axioms, one of which is an atom, the other - negation of this atom.

The program introduced below has 3 types of variables, 4 basic relations T , $T0$, $Tree$, $Proof$ and 8 additional relations $W1 - W4$, $Pair$, $Resolution$, $Several$ and $Exist$.

We use the next denotations for variables:

A_0, A_1, A_2, \dots - variables with the value equal to codes of the axioms, below these variables are called just axioms;

a_0, a_1, a_2, \dots - variables with the value equal to codes of the atoms, below these variables are called just atoms;

b_0, b_1, b_2, \dots - variables with the value equal to signs of atoms (0 for atoms without negation or 1 for atoms with negation).

The basic relations contain the following information:

- $T(A_1, a_1, b_1)$ is a relation for a theory, presumptuously inconsistent. This relation for each axiom A_1 points at the atoms a_1 present in the axiom, and the signs of these atoms.

- $T0(A_1, a_1, b_1)$ is a relation including T and all resolvents received during iteration.

- $Tree(a_1, A_1, A_2, A_3, A_4)$ is a tree of deduction. In this relation the atom a_1 is eliminated at a given stage of iteration, A_1 and A_2 are codes of the axioms, taking part in the resolutions, A_3 is a maximal code of the axioms to the beginning of a given stage of iteration, A_4 is maximal code of the axioms during execution of the given stage ($A_4 \geq A_3$). The code A_3 is used for scanning of all pairs of axioms taking part in the resolutions of a given stage ($A_1 < A_2 \leq A_3$). New axioms (resolvents) are not used at scanning (at the given stage). The code A_4 is used for coding of new axioms (the code of a new axiom equals $A_4 + 1$). After coding we increase the value A_4 by 1.

- $Proof(A_1, A_2, A_3)$ contains the list of resolutions leading to an empty resolvent.

The additional relations contain the following information:

- $W1(A_1, A_2)$ includes axioms A_2 not subsumed by A_1 , i.e. the axioms A_2 contain atoms missing in A_1 or present in A_1 , but with a different sign. Negation of the relation $W1$ includes A_2 subsumed by A_1 .

- $W2(A_1)$ contains all subsumed axioms.

- $W3(A_1, a_1, a_2)$ sets the order of atoms ($a_1 < a_2$) in the axiom A_1 . The relation $W3$ is necessary for finding a maximal atom in the axiom. This atom cannot be a_1 .

- $Pair(a_1, A_1, A_2)$ selects the axioms A_1 and A_2 with a contrary pair of a maximal (in both axioms) atom a_1 . Except this contrary pair the axiom A_1 and A_2 can additionally contain some contrary pairs.

- $W4(a_1, A_1, A_2)$ selects in relation $Pair$ axioms A_1 and A_2 with several contrary pairs, one of these pairs includes the atom a_1 . Negation of $W4$ includes axioms with a single contrary pair including a maximal atom a_1 .

- $Resolution(A_1, A_2, a_1)$ selects in the relation $Pair$ axioms A_1 and A_2 not subsumed by other axioms and with a single contrary pair including the maximal atom a_1 .

- $Several(A_1)$ is a list of axioms consisting of two and more atoms. If a resolution contains an axiom from this list as an argument then the resolvent cannot be empty.

- $Exist(a_1, A_1, A_2)$ establishes the resolution of the axioms A_1 and A_2 has a nonempty resolvent. The atom a_1 is maximal in both axioms and forms a contrary pair.

The relations $W1 - W4$ allow to remove universal quantifiers, so these relations meet only with negation in the right part of sentences of the program.

Next we give the text of the program with commentaries.

All additional relations are calculated by rules (1)-(9):

$$W1(A_1, A_2) \leftarrow T0(A_2, a_1, b_1) \wedge \neg T0(A_1, a_1, b_1) \quad (1)$$

$$W2(A_1) \leftarrow \neg W1(A_1, A_2) \quad (2)$$

$$W3(A_1, a_1, a_2) \leftarrow T0(A_1, a_1,) \wedge T0(A_1, a_2,) \wedge a_1 < a_2 \quad (3)$$

$$Pair(a_1, A_1, A_2) \leftarrow T0(A_1, a_1, b_1) \wedge T0(A_2, a_1, b_2) \wedge b_1 \neq b_2 \wedge \neg W3(A_1, a_1,) \wedge \neg W3(A_2, a_1,) \quad (4)$$

$$W4(a_1, A_1, A_2) \leftarrow Pair(a_1, A_1, A_2) \wedge T0(A_1, a_2, b_1) \wedge T0(A_2, a_2, b_2) \wedge a_1 \neq a_2 \wedge b_1 \neq b_2 \quad (5)$$

$$Resolution(A_1, A_2, a_1) \leftarrow Pair(a_1, A_1, A_2) \wedge \neg W4(a_1, A_1, A_2) \wedge \neg W2(A_1) \wedge \neg W2(A_2) \quad (6)$$

$$Several(A_1) \leftarrow T0(A_1, a_1,) \wedge T0(A_1, a_2,) \wedge a_1 \neq a_2 \quad (7)$$

$$Exist(a_1, A_1, A_2) \leftarrow Resolution(a_1, A_1, A_2) \wedge Several(A_1) \quad (8)$$

$$Exist(a_1, A_1, A_2) \leftarrow Resolution(a_1, A_1, A_2) \wedge Several(A_2) \quad (9)$$

In rule (10) relation $T0$ is initialized by a relation T , in rules (11) and (12) this relation includes one more resolvent received during constructing of deduction tree $Tree$. This resolvent has code A_1 , extracted from $Tree$, atoms of the resolvent and their signs are extracted from the axioms taken part in the resolution. By the rule (11) the atoms and the signs are extracted from axiom with less code, by the rule (12) - from axiom with greater code.

$$T0(A_1, a_1, b_1) \leftarrow T(A_1, a_1, b_1) \quad (10)$$

$$T0(A'_3, a_2, b_1) \leftarrow T0(A_1, a_2, b_1) \wedge Tree(a_1, A_1, A_2, , A_3) \wedge Exist(a_1, A_1, A_2) \wedge a_1 \neq a_2 \quad (11)$$

$$T0(A'_3, a_2, b_1) \leftarrow T0(A_2, a_2, b_1) \wedge Tree(a_1, A_1, A_2, , A_3) \wedge Exist(a_1, A_1, A_2) \wedge a_1 \neq a_2 \quad (12)$$

By the rule (13) relation $Tree$ is initialized by:

- a maximal atom,

- axioms with code 0 and 1,

- a maximal code of the axioms in the relation T .

$$Tree(a_1, 0, 1, A_1, A_1) \leftarrow T(A_1, ,) \wedge \neg T(A'_1, ,) \wedge T(, a_1,) \wedge \neg T(, a'_1,) \quad (13)$$

In the following iterations the resolvent of the axioms A_1 and A_2 is sought. If the resolution of these axioms is impossible, the code of the second axiom is increased by 1:

$$Tree(a_1, A_1, A'_2, A_3, A_4) \leftarrow Tree(a_1, A_1, A_2, A_3, A_4) \wedge A_2 \leq A_3 \wedge \neg Resolution(a_1, A_1, A_2) \quad (14)$$

If the resolution is possible and the resolvent is nonempty, the maximal code of the axioms A_4 is increased by 1 too and assigned to a new resolvent (rule 15). If the resolution is possible but the resolvent is empty, the iteration is ended (rule 16). For blocking further iteration a_1 is set equal to 0, A_1 is set equal to the maximal code of the axioms, A_2 is set equal to a number of the axioms plus 1, A_3 and A_4 are set equal to a number of the axioms (a number of the axioms by 1 greater than a maximal code of the axioms):

$$Tree(a_1, A_1, A'_2, A_3, A'_4) \leftarrow Tree(a_1, A_1, A_2, A_3, A_4) \wedge A_2 \leq A_3 \wedge Exist(a_1, A_1, A_2, A_4) \quad (15)$$

$$Tree(0, A_4, A''_4, A'_4, A'_4) \leftarrow Tree(a_1, A_1, A_2, A_3, A_4) \wedge A_2 \leq A_3 \wedge Resolution(a_1, A_1, A_2) \wedge \neg Exist(a_1, A_1, A_2, A_4) \quad (16)$$

If all codes for the second axiom are exhausted, the code of the first axiom is increased by 1 and the code of the second axiom becomes by 1 greater than a new code of the first axiom:

$$Tree(a_1, A'_1, A''_1, A_3, A'_4) \leftarrow Tree(a_1, A_1, A'_3, A_3, A_4) \wedge A'_1 \neq A_3 \quad (17)$$

If all codes for both axioms are exhausted, we go to a new stage. The value of a_1 is decreased by 1, the value of A_1 equals 0, the value of A_2 equals 1, the values of A_3 and A_4 equal the maximal code of the axioms:

$$Tree(a_1, 0, 1, A'_4, A'_4) \leftarrow Tree(a'_1, A_1, A''_1, A'_1, A_4) \quad (18)$$

The relation $Proof(A_1, A_2, A_3)$ is initialized by the resolution of monatomic axioms. For these axioms the resolution is possible, but a resolvent does not exist. The value of A_3 , taken from the relation $Tree$, equals a maximal number of the axioms obtained during the iteration. This value is unique, as the maximal code of the axioms is less than a number of the axioms by 1:

$$Proof(A_1, A_2, A_3) \leftarrow Resolution(, A_1, A_2) \wedge \neg Exist(, A_1, A_2) \wedge Tree(0, , A'_3, A_3, A_3) \quad (19)$$

Further, the resolutions are sought in the relation $Tree$. Resolvents of this resolution must be

equal to A_1 (rule 20) or A_2 (rule 21). For each of found resolutions we search the next resolutions. These resolutions must have resolvents being arguments of the previous resolutions from *Proof*:

$$Proof(A_1, A_2, A'_3) \leftarrow Proof(A'_3,) \wedge Tree(, A_1, A_2, , A_3) \quad (20)$$

$$Proof(A_1, A_2, A'_3) \leftarrow Proof(, A'_3,) \wedge Tree(, A_1, A_2, , A_3) \quad (21)$$

The execution of the program is finished. The obtained list of the resolutions is ordered on ascending of codes of resolvents. Then this list becomes to be the proof of the source theorem by contradiction.

The sequence of the execution of rules (1) - (21) is arbitrary, but effectiveness of the rules becomes very high if we put the order in the set of these rules. As it is known [1], this order can be automatically put.

But the ordering of relations (to be exact, of sets interpreting these relations) increases effectiveness of calculations too. Rules of such ordering generally do not exist yet. Therefore this program can be used only as formalization of the rule of sequential removing of atoms.

To increase effectiveness we can recommend the following rule of ordering of the relation $T0$.

The axioms in this relation $T0$ are broken down into 3 groups:

- the axioms with the maximal given atom without negation;
- the axioms with the maximal given atom with negation;
- the other axioms.

The atoms inside the axioms (in the first two groups) are ordered on ascending signs, at identical signs - on descending codes of atoms. The axioms inside the group are ordered lexically: on the ascending sign of their first atoms, at identical signs - on descending codes of these atoms, at identical signs and codes of the first atom - on ascending the sign of the second atoms and so on.

At ordering all subsumed axioms are removed.

In the third group the arbitrary order is put.

Other types of ordering inside groups are also possible.

The atom, used for dividing axioms into 3 groups, is selected in the beginning of each stage of iteration. A number of stages equals a number of atoms in the theory. Therefore a number of orderings is equal to a number of atoms in the theory.

So in relational programming a compiler can construct effective programs only if this compiler has high intellect. This intellect is needed for automatic ordering of tables as basic tools to increase effectiveness of programs. It is a very difficult problem. Therefore intellect of a compiler should be as good as man intellect.

Before creating such compiler we can recommend to use relational programming only for formalizing logical rules or for solution of simple problems.

References

- [1] M.A. Malkov. Introduction to relational logic. *Relational logic*, 2001, 1.
- [2] J. Lukasiewicz. The equivalential calculus *Jan Lukasiewicz: Selected Works*, North-Holland, Amsterdam (1970).
- [3] L. Wos. Searching for cycles of pure proofs. *J. Automated reasoning*, 15, 1-29 (1995).
- [4] L. Wos. Meeting the challenge of fifty years of logic. *J. Automated reasoning*, 6, 213-232 (1995).
- [5] J. Kalman. A shortest single axiom for the classical equivalential calculus, *Notre Dame J. formal logic*, 19, 141-144 (1978).
- [6] L. Wos, S. Winker, R. Veroff, B. Smith and L. Henschen. A new use of an automated reasoning assistant: Open questions in equivalential calculus and the study of infinite domains. *Artificial intelligence*, 22, 303-356 (1984).
- [7] A. Marien. Private communication (1984).
- [8] D. Maier. *The theory of relational databases*, Computer science press, 1983.
- [9] M.A. Malkov. Relational propositional logic. *Relational logic*, 2001, 1.