Andrzej Biela

Algorithmic structural completeness and a retrieval system for proving theorems in algorithmic theories

Wydawnictwo Uniwersytetu Śląskiego

Katowice 2000

Redaktor serii: Matematyka
Tomasz Dłotko

Recenzenci
Andrzej Salwicki
Andrzej Skowron

Contents

Chapter 1

Introduction

The paper presents the proofs of algorithmic formulas. These formulas, as a part of the language of algorithmic logic, make it possible to express:

- the properties of programs e.g. correctness,
- the definitions of semantics of language of programming,
- the data structures e.g. the structures of trees, stacks, etc.

Only the proof of formula expressing the correctness of the program in relation to the proper specification can assure the user of error-free application of the program.

However, some advanced tools and programming languages the correctness of programs is not always easy to verify. Therefore, in this paper we use the method of proving called *abduction*. In consequence it is possible to obtain the value of function by means of the proof. This technique was first mentioned by Herbrand in his definition of recursive function.

This paper consists of two parts:

- research on algorithmic structural completeness of algorithmic logic,
- description of the retrieval system RS providing comprehensive tools in automated theorem proving theorems of algorithmic theories i.e. theories based on algorithmic logic,

  description of the RETRPROV system which enables us to prove theorems of algorithmic theories by using the decomposition rules.

In the first part of our paper (cf. Chapter 3) we present a possibility of introducing the notion of program-substitution as a special mapping from the set of all formulas of AL into the same set.

Let $\mathcal{N}$ be the set of non-negative integers. In AL we have some kind of substitution i.e. an assignment instruction s of the form:

$$[x_1/\tau_1, \ldots, x_n/\tau_n, a_1/\alpha_1, \ldots, a_m/\alpha_m]$$

for $n, m \in \mathcal{N}$, where $x_1, \ldots, x_n$ (respectively $a_1, \ldots, a_m$ denote pairwise different individual (respectively propositional) variables, $\tau_1, \ldots, \tau_n$ are classical terms, $\alpha_1, \ldots, \alpha_m$ are classical open formulas and for example x_1/τ_1 means the standard assignment instruction $x_1 := \tau_1$.

Unfortunately this form of substitution s in the formula $\overline{s\rho(x_1, \ldots, x_n)}$ transforms the formula $\rho(x_1, \ldots, x_n)$ only into the formula $\rho(\tau_1, \ldots, \tau_n)$ but not for example into the formula of the form $\alpha \wedge \beta$ where $\wedge$ denotes the conjunction.

Our substitution called program-substitution has not this restriction so it is more general.

Various attempts have been done to introduce the substitution rule in any logical systems by A. Church [18], H. Hermes [42], D. Hilbert [43] and W. A. Pogorzelski and T. Prucnal [71] but our key idea slightly differs from the methods developed up to now because contrary to the previous substitution rules it does preserve the properties of programs.

In Chapters 2, 3, 4 we define the notions of the consequence operation, the admissible, finitary and derivable rules which enable us to introduce the notion of algorithmic structural completeness and to prove that the consequence operation of algorithmic logic strengthened by the substitution rule is algorithmically structurally complete i.e. that every structural finitary and admissible rule is derivable in this consequence operation. This result gives us the useful class of rules.

The notion of structural completeness of a logical system was introduced by W. A. Pogorzelski [70] and thoroughly studied in propositional logics as well as in the systems with quantifiers by A. Biela [4], A. Biela and W. Dziobiak [6], M. Dummet [25], G. Minc [57], W. A. Pogorzelski and T. Prucnal [71], J. Porte [72], T. Prucnal [76], A. I. Tsitkin [98] and P. Wojtylak [104].

In the first part of this paper we shall prove that the consequence operation C_{R_*} of algorithmic logic strengthened by the substitution rule is algorithmically structurally complete though it is not complete.

Here we explain the notion of algorithmic structural completeness of a logic of programs which plays an important role in this paper.

In our paper the definition of the consequence operation C_{R_*} of algorithmic logic strengthened by the substitution rule will be based on the set of axioms Ax and on the set of rules R of algorithmic logic. Thus $R_* = R \cup \{r_*\}$.

The purpose of this work is to show a point of view upon the notions of program-substitution and admissibility of rules which are the tools for proving properties of programs in algorithmic logic and in the so-called extended algorithmic logic with quantifiers and with non-deterministic programs. We shall prove that these logics are closed under each program-substitution i.e. $p(C_{R_*}(\emptyset)) \subset C_{R_*}(\emptyset)$ for every program-substitution p.

As we mentioned above the consequence operation C_{R_*} of algorithmic logic is not complete, so it is not true that each admissible rule of C_{R_*} is derivable in C_{R_*}. Therefore we looked for a weaker kind of the notion of completeness. We tried strengthening the notion of the substitution rule r_* to get the following property: each structural, finitary and admissible rule in C_{R_*} is derivable in it.

This property called algorithmical structural completeness means that every consequence operation which has this property is intuitively quasi-complete i.e. it is complete because of structural, finitary and admissible rules. Since AL is algorithmically structurally complete thus we can use every structural and admissible rule in C_{R_*} while proving theorems of AL which simplifies the proof.

Chapter 2 begins with the definition of the language of AL. There we develop a formal model theory of AL. This Chapter contains a formal system for AL and the consequence operation of this system. In Chapter 3 we define the set of the program-substitutions and we proof that AL is closed under program-substitution. Moreover Chapter 3 contains a proof that any program-substitution preserves the logical connectives. In Chapter 4 we proved the algorithmic structural completeness of the consequence operation of algorithmic logic strengthened by the substitution rule as well as its incompleteness. This chapter contains some remarks about program-substitution in AL with generalized terms and with quantifiers and with non-deterministic programs.

The second part of this paper i.e. Chapter 5 and Chapter 6 presents a retrieval system (RS-algorithm) investigated by A. Biela [5] and a decomposition system described by A. Biela and J. Borowczyk [7] in which the properties of programs are expressed.

Further in this paper we shall describe a formal system which enables us to prove theorems from the following theories: propositional calculus, logic of quantifiers and the first-order theories. However, the theories of algorithmic logic including theorems containing programs are the most important ones. Its main feature relies on generating an additional set of assumptions needed to prove a considered formula. Thus we are able to consider expressions which can become theorems by adding the special set of assumptions (axioms) to the standard set of axioms. RS-algorithm is looking for a special set of axioms to prove the considered formula.

We shall try to show some methods and procedures investigated by A. Biela [5] for constructing formal proofs of theorems of algorithmic logic containing programs.

Our methods concern proving by means of programming. They are an essential extension of methods used by P. Gburzyński [28], [29]. The considered retrieval system is able to solve or to prove:

1. the properties of programs and terms formulated in the language of arithmetic,
2. the correctness of some programs with STOP property,
3. the functional equations with the recursive functions defined by programs. This system solves them in a dynamic way by looking for a special set of axioms during the execution of algorithm,
4. the relations defined by programs and recursive functions,
5. the equivalence of programs.

Therefore we can answer whether some relations hold and we are able to compare programs and get an answer, whether the execution of different programs gives the same result. At the end of this section we present some experimental results.

Though the solution to considered problems are very ineffective, the options and methods used by us are satisfactory in practice (see Table 1 of experimental results).

Our proposal has in view:

1. to provide the tools for didactics, which enable us to demonstrate on the monitor the proofs of theorems of the calculus of quantifiers, algorithmic logic, algorithmic theories, propositional calculus, geometry, set theory, theory of lattice, boolean algebra...,
2. to enable us to undertake a trial of proving hypotheses,
3. to secure the specific results for example the independence of axioms,
4. to verify the correctness of definitions,
5. to verify some hypothesis.

The retrieval system can be used for giving an expert appraisement because it works in a broad area and can solve different problems, so it is an expert system.

We believe, that it is reasonable to use in our considerations some formalism of the language of algorithmic logic described by L. Banachowski [1], G. Mirkowska [58], [59], G. Mirkowska and A. Salwicki [64], A. Salwicki [87] and H. Rasiowa [82]. The language of algorithmic logic contains all classical formulas and generalized formulas describing properties of algorithms which can be interpreted in our considerations in a model of arithmetic or in a model of integers.

The properties of algorithms from the point of view of recursion theory and degree of undecidability of algorithmic properties were settled by W. Dańko [20], [21] and A. Kreczmar [48].

The main idea depends on handling the expressions of the forms $K\tau$-generalized term and $K\alpha$-generalized formula where K is a program, τ is a generalized term or a classical term and α is a classical or a generalized formula. These expressions enable us to describe functions or relations defined by programs and recursive functions. For example the factorial $n!$ can be defined in algorithmic logic by a generalized term of the form $K_1 z$ where:

K_1: **if** n = 0 **then** $z := 1$ **else** $z := \mathrm{n}*\mathrm{f}(\mathrm{n}-1)$;,

for $f(\mathrm{n}) = K_1 z$, while the order relation between natural numbers can be expressed in algorithmic logic by generalized formula of the form $K_2 a$, where

```
K₂: if x = y then a := FALSE else
      begin u := 0;
            while ¬((u = y) ∨ (u = x))
                  do u := u + 1;
            if u = x then a := TRUE else a := FALSE;
      end;
```

Readers accustomed to formalism of Hoare should observe certain difference in semantics. We can show this difference by giving a typical example. For example the expression $K\tau = u$ can be sensibly considered even when u does not occur in τ.

Now we explain a technique which by means of a proof enables us to get information about the value of function. The considered function will be defined by a program. The pioneer of this method (called ABDUCTION) mentioned in the definition of recursive functions was J. J. Herbrand.

Let us consider the definition of factorial $f(\mathrm{n}) = K_1 z$. If we consider the expression $f(3) = u$ availing itself of the definition of function f, given by the program K_1, then the equality $u = 6$ is the result of our system. This obtained equality may be interpreted as a question whether $f(3) = u$ is a theorem under the assumption $u = 6$. Our system will find the equality $u = 6$ and it will use it to prove the equality $f(3) = u$. On the one hand the number 6 in the equality $u = 6$, may be interpreted as a result of calculation of $K_1 z$, on the other hand the equality $u = 6$ may be interpreted as a special axiom in the proof of the equality $f(3) = u$.

In our considerations only the second interpretation is suitable. To show the difference between the proof of $u = f(3)$ and the calculation of $K_1 z$, which is used for changing $f(3)$ by the result of this calculation, we consider the program defining the addition:

> K_3: **if** $y = 0$ **then** $z := x$ **else** $z := k(x, y - 1) + 1$; where
> $k(x, y) = K_3 z$.

If we want to prove the equality $k(x, 1) = u$, the retrieval system needs the equality $u = x + 1$ during the proof. Obviously, the calculation of every program realizing the addition function in the set of integers gives us as the result the number and not the expression of the form $x + 1$.

In the same way our system gives us the answer whether some relation holds or not. Let us consider the relation $\rho(x, y) \equiv K_2 a$. If we want to get the answer whether $\rho(1, 2)$ holds or not, our system will attempt to prove the expression of the form $\rho(x, y) \equiv b$. During the proof it gets the answer that $b \equiv$ TRUE.

We shall give the main idea of this algorithm. If we want to prove a classical formula without functions and relations defined by programs then our algorithm gives us the proof in a standard way. It uses the rules to decompose sequents i.e. the expressions of the form $X \Vdash Y$, where X and Y are two sequents of generalized formulas. If the constructed diagram of the considered classical formula is finite and all leaves are axioms then we get the proof of this formula. But when we want to prove an expression containing a function or relation defined by program then to explain this algorithm we take for example the formula $\varphi(t_1, \ldots, t_n) = Mt$. Thus Mt can be treated as the definition of the function $\varphi(t_1, \ldots, t_n)$. Our algorithm starts with the sequent of the form: $\Vdash \varphi(t_1, \ldots, t_n) = u$. Next we change the function by its definition Mt, so we get the sequent of the form $\Vdash Mt = u$. After that we move the program M outside the equality and we get $\Vdash M(t = u)$. Next we use the rules to decompose the program M and we do it up to the moment, when we get the sequent composed only of the classical formulas from At. If such obtained sequent has on the right side of the symbol $\Vdash$ only one classical formula of the form $\tau = u$ (where, intuitively saying τ is the result of the execution of the program M on the term t) then we extend the set of axioms by adding the set of special axioms i.e. sequents containing $u = \tau$ on the right side of the symbol $\Vdash$. Such an operation enables us to get the proof of the classical formula $\Vdash \varphi(t_1, \ldots, t_n) = u$ by our system.

We explained the idea of the execution of the considered system and we showed how during the proof we ought to choose the special set of axioms.

Now we explain the activity and the usage of the retrieval system. The idea of working of this system avails itself of conception of resolution and Gentzen's method. To realize our conception of looking for the axioms we introduce many options. Moreover the decomposition of the program **while** α **do** K requires a special treatment.

Let us consider the following definitions:

$f(n) = K_1 z$
$\rho(x, y) \equiv K_2 a,$
$k(x, y) = K_3 z,$
$g(x) = K_4 z,$ where K_4 is of the form **begin** $i := i + 3$; $z := x$ **end**,
$h(x, y) = K_5 z,$ where K_5 is of the form **if** $x = 0$ **then** $z := 2$ **else**
$z := h(x - 1, h(x, y)).$

By the above definitions our system will try to prove the following properties:

$$f(1) = u,\ \rho(1,2) \equiv b,\ k(x, 1) = u_1,\ g(n^4) = u_2,\ h(1, 2) = u_3.$$

The environment of our system consists of two sets DEF and DAT. In DEF we write the definitions which are needed during the proof of considered expression. In DAT we put the formula which our system will try to prove. Using the above mentioned classical formulas we shall give the graphic illustration of execution of our system:

ENVIRONMENT	
DEF	DAT
$f(n) = K_1 z$	$f(1) = u$

ENVIRONMENT	
DEF	DAT
$\rho(x, y) \equiv K_2 a$	$\rho(1, 2) \equiv b$

ENVIRONMENT	
DEF	DAT
$k(x, y) = K_3 z$	$k(x, 1) = u_1$

After using the definition the retrieval system will try to prove the following expressions:

$n := 1(K_1 z = u)$	$x := 1(y := 2(K_2 a \equiv b))$	$y := 1(K_3 z = u_1)$

Further execution of the retrieval system:

Paragraph 5.3	Paragraph 5.4	Paragraph 5.4
Example 7	Example 10(*iv*)	Example 10(*iii*)

Our system finds the additional premises which enable us to prove the above properties:

$u = 1$	$b \equiv$ TRUE	$u_1 = x + 1$

Fig. 1

ENVIRONMENT	
DEF	DAT
$g(x) = K_4 z$	$g(n^4) = u_2$

ENVIRONMENT	
DEF	DAT
$h(x, y) = K_5 z$	$h(1, 2) = u_3$

After using the definition the retrieval system will try to prove the following expressions:

$x := n^4(K_4 z = u_2)$ $\qquad$ $x := 1(y := 2(K_5 z = u_3))$

Further execution of the retrieval system:

Paragraph 5.4 $\qquad$ Paragraph 5.4

Example 10(i) $\qquad$ Example 10(ii)

Our system finds the additional premises which enable us to prove the above properties:

$u_2 = n^4$ $\qquad$ $u_3 = 2$

Fig. 2

We have to mention that by the retrieval system we can verify the correctness of programs. To explain it let us consider the program defining the factorial i.e. $f(n) = K_1 z$ (instead of K_1 we can consider another program defining the factorial). If we want to get the answer whether K_1 is well written i.e. whether the program K_1 really defines the factorial (for every natural number n), we need to prove the expression of the form:

$$f(0) = 1 \wedge \forall_x(\neg(x = 0) \rightarrow f(x) = x * f(x - 1))$$

because only the factorial fulfils this recursive condition. So a program defining a recursive function can be verified in such a manner.

The graphic illustration of the proof of correctness of program defining the factorial is as follows:

ENVIRONMENT	
DEF	DAT
$f(n) = K_1 z$	$f(0) = 1 \wedge \forall_x(\neg(x = 0) \rightarrow f(x) = x * f(x - 1))$

After using the definition the retrieval system will try to prove the following expression:

$$n := 0(K_1 z) = 1 \wedge \forall_x(\neg(x = 0) \rightarrow (n := x(K_1 z) = x * (n := (x - 1)(K_1 z))))$$

The retrieval system will prove the above generalized formula.

Fig. 3

The above considered example as well as the others were tested and we present the time of execution (see paragraph 5.6, Table 1).

The above presented examples show that the constructed algorithm computes even such generalized formulas for which the standard computation is helpless since it cannot compile the program defining the function $h(x, y)$. However the retrieval system will be able to get the result.

In this paper we shall provide the major structures of the implementation. The generalized terms, formulas and programs are represented by the object TNODE consisting of four fields. Two fields are for the name of individual or propositional variable, logical constant, generalized quantifier, iteration quantifier, logical connectives and program connectives. The next two fields are the pointers of the same type as the considered object. The sequent is represented by the object consisting of two fields of the type TNODE and one field being a pointer to the object of the type of SEQUENT. These objects enable us to program the algorithm of retrieval system (RS-algorithm).

When we consider the correctness of the program defining the factorial we can see that our system is able not only to prove the equalities of the form $\varphi(t_1, ..., t_n) = u$ or to verify the relations, but also it can prove the generalized formula from algorithmic logic. As an example we can prove the expression of the form:

$$x > 2 \rightarrow (f(x) = (x * (x - 1) * f(x - 2))).$$

All these possibilities are expressed in the language of algorithmic logic where the expressions $\varphi(t_1, ..., t_n)$ (i.e. recursive functions) can be defined by generalized terms of the form $K\tau$. Moreover we can prove or verify by RS-algorithm the expressions from many theories. For example we shall formulate some of them:

1. If x is a finite set and $y \subset x$ then the power of the set y is less than the power of the set x, for every set x and y (it is a theorem of set theory),
2. If $T(x, y, z, v)$ is a trapezium then the angles zyv and zvy are equal (it is a theorem of geometry),
3. $(P(x) \rightarrow \forall_x Q(x)) \equiv \forall_y (P(x) \rightarrow Q(y))$ (it is a theorem of the calculus of quantifiers),
4. $\neg(\exists_x P(x) \vee \exists_y Q(y)) \vee \exists_z (P(z) \vee Q(r))$ (it is not a theorem of the calculus of quantifiers),
5. $(p \rightarrow (q \rightarrow s)) \rightarrow ((p \rightarrow q) \rightarrow (p \rightarrow s))$ (it is a theorem of propositional logic),
6. $(X \cup Y) \setminus Z = (X \setminus Z) \cup (Y \setminus Z)$ (it is a theorem of boolean algebra),
7. $\forall_x((\forall_y x \cup y = y \rightarrow x = 0) \wedge (\forall_y x \cap y = y \rightarrow x = 1))$ (it is a theorem of lattice).

We shall construct an expert system which will be able to solve problems in a similar way to the human brain. The procedures and functions may occur in the considered theorems while the program is being executed.

In the last section we shall study the decomposition of programs by using the model and we shall describe two rules which play an essential role in our considerations. In this section we shall formulate the **RETRPROV**-algorithm which enable us to prove theorems as well as to find a special set of axioms for expressions containing procedures and functions defined by programs. Some Gentzen method was considered by G. Mirkowska [61] and by A. Kolany in his manuscript. We shall not use the Gentzen's method but by a special kind of decomposition we shall get the result in an evidently shorter and speedier way than by using **RS**-algorithm. We shall present a few examples of using **RETRPROV**-algorithm for proving properties of programs. **RETRPROV**-algorithm enables us to determine whether a relation defined by program holds. Moreover it can be applied to Hoare's method for proving partial correctness of programs. If M is a program, α is a generalized formula, and β is an output generalized formula then the problem of partial correctness of program M can be reduced to the question of whether the formula $(\alpha \wedge M\ TRUE) \rightarrow M\beta)$ is true.

Chapter 7 contains the concluding remarks and the summary of the author's contribution to automatic proving system.

Historical remarks

The starting-point of our considerations was an idea connected with functions defined by programs which was mentioned by A. Salwicki [89].

Today there exist many systems formalizing the mathematical semantics of programming languages. In the presented paper we consider a logical system in which the properties of programs are expressed. This logical system called *algorithmic logic* AL was initiated by A. Salwicki [88] in 1970. It includes expressions called programs and generalized formulas describing properties of programs. *Programs* are expressions built by means of substitutions as *primitive programs* being interpreted as assignment statements and by means of operations of composition, branching and iteration. These correspond to basic operations in programs written in high level languages such as **FORTRAN**, **ALGOL** or **PASCAL**. In that way an algebra of programs was obtained. This was not the aim in itself but an auxiliary step in the development of theory.

At first the axiomatizability of algorithmic logic was established by L. Banachowski [1], W. Dańko [21] and G. Mirkowska [58], [59], [60], [62], next the questions of effectivity problems of AL were studied by B. Chlebus

[17] and A. Kreczmar [48], [49]. Moreover many-valued algorithmic logic were considered by E. Perkowska [68] and H. Rasiowa [80]. Applications of algorithmic logic to procedures have been discussed by S. Radziszewski [77], H. Rasiowa [82] and others. Some logical systems enable us to examine *non-deterministic algorithms*. They are related to the *dynamic logic* formulated by V. Pratt [73] and investigated in several papers by D. Harel and V. Pratt [39] and K. Segerberg [90] as well as by G. Mirkowska [60], [61] in *algorithmic logic with non-deterministic programs*.

H. Thiele [96] and E. Engeler [27] were that first who were looking for formalized logical systems dealing with programs and their properties.

The history of automated deduction described in the literature is very extensive. Nowadays there are two methods often applied in automated theorem proving i.e. resolution which was studied by J. Robinson [83] and C. Green [33] and G. Gentzen's method [30].

Robinson used resolution for the first-order logic and showed its practical use. In fact, it is correct to say that all details connected with resolution were known before J. Robinson [83]. Resolution as a propositional rule was defined as a function and studied by A. Blake [9].

Next it became well-known as Quine's consensus rule [79] of the form:

$$\frac{\Vdash p; p \Vdash q}{\Vdash q}$$

which in turn is just a variant of Gentzen's cut rule [30]

$$\frac{X \Vdash Y, p; p, Z \Vdash W}{X, Z \Vdash Y, W}$$

and is the generalized version of the modus ponens. Gentzen's method is competitive to all methods using resolution as a main rule (see M. Davis [23]).

B. Dunham and J. North [26] used the consensus rule in a version of W. V. Quine as a recognition-type rule for theorem proving. Unification, however, was first discovered by J. J. Herbrand [41] and used by D. Prawitz [74].

Robinson's achievements consisted in putting all these results together into a uniform and elegant calculus [84].

The linear refinement of resolution was introduced independently by D. W. Loveland ([52], [51]) and by D. Luckham [54].

Detailed comparisons of different proof procedures in the linear strategy were carried out by G. V. Davydov [24], D. W. Loveland [53], W. Bibel [3], W. Chang and L. Lee [16] and by D. W. Loveland [50].

The first implementation of a proof procedure for the first-order logic was done by D. Prawitz, H. Prawitz and N. Voghera in 1958/59 [75] and the first implementations of mathematical theorem proofs were done in the midfifties. For instance, in 1954 M. Davis [23] implemented Presburger's decision procedure for the arithmetic addition.

In 1956 A. Newell, J. Show and H. Simon [65] constructed a program called the *logic theorist* for proving theorems in propositional logic in a way which simulated the human problem solver.

In the world literature we can find a review of various proving systems e.g. R. S. Boyer and J. S. Moore [12] present one of them, which verifies the properties of recursive functions. This system employs the reduction and induction. Some lemmas in Boyer and Moore's interactive proving system are specified to be proved before their using in the main theorem.

Several heuristics make the proving theorems more general. This is an incomplete system. The heuristics enhance its effectiveness. This system verifies programs and theorems of mathematics and metamathematics (A. Bundy [14]), as well as Wilson's theorem (D. Rusinoff [86]).

L. M. Hines's system of proving theorems [45], [44] transforms several simple conclusions into more general ones which simplifies concluding due to elimination of auxiliary results.

In consequence the usage of these rules is bounded which, however, does not detract from the value of the results or accelerates the proving process.

The next proving system constructed by S. A. Miller and L. K. Schubert [56] recognizes natural language. It is a hybrid system namely a resolution proving system equipped with the specialized concluding modules concentrating on the fixed data structure which accelerates concluding. In this system there are modules calculating in the arithmetic theory and the set theory. This module was described by J. Haan and L. K. Schubert [38]. The theorems which are proved by this system are formulated in the language of the first-order predicate calculus.

H. S. Jonsohn, R. Landwehr, G. Writson [46], [47] present an interactive proving system based on J. A. Robinson's solution [85]. This system accepts expressions of lambda calculus. This system applies semantic approach in generating proofs by contradiction.

The next system constructed by S. Greenbaum [34], [35] uses various variants of the resolution method. Complex data structures make it possible to avoid redundancy which results from storing a lot of copies of the same objects and to accelerate the search of required information from a data base.

M. E. Stickel's system [94], [93], [92], [91] is based on the resolution method represented by graph. The formulas including the equality symbol are simplified by means of a reduction system. One element of this system is a prologlike proving system.

M. Gordon, A. Milner and C. Wadsworth [32] and M. J. C. Gordon [31] tested LCF program which verifies the properties of calculable functions defined in the language of the first-order predicate calculus and lambda calculus. The strategies of theorem proving were formulated by L. Cardelli in user-friendly programming language ML [15]. This system was applied in testing several standard mathematical theorems. It was also tested by L. Paulson [67].

E. L. Lusk, W. W. McCune and R. A. Overbeek [55] constructed programs which enable the user to apply many functions from different proving systems. These programs are convenient to use. The elements of this system are grouped into five levels.

- In the first level there are several implementations of primitive types nonexistent in Pascal.
- In the second level the type "object" was implemented. On the elements of the type "object" we can use the unification and the substitution rule. In this level there are mechanisms allowing to represent and to use logical formulas and then substitution. Moreover each object can have some attributes.
- On the third level we can use functions allowing to conclude by resolution and to absorb clausules.
- On the fourth level it is possible to do a configuration of the whole proving systems. The systems on this level are represented by independent processes.
- On the fifth level the modules are able to manage the processes from the fourth level. The tools in this system are general enough to construct the proving theorem system in lambda calculus, the system of natural deduction, Gentzen's system and the system based on the resolution rule.

K. M. G. Raph [78], K. Blasius, N. Eisinger, J. Siekmann, G. Smolka, A. Herold, C. Walter [10], A. Bundy [14] and H. J. Olbach [66] constructed the system in Kaiserslautern and Karlsruhe which belongs to the greatest projects of this type. This system assumes that proving theorems requires extensive, specific knowledge which is used to formulate theorems. It consists of two levels:

- The aim of the first level is to gather information (axioms, definitions etc.) which is specific for the considered problem and to decide about the way of proving. Moreover this level chooses the proper strategy and makes the suitable modules active.
- The second level is based on the structure of graph in which each edge is a potential step in concluding in the set of clausules e.g. the edge of the graph means using resolution or factorization. Concluding is possible because of special modules. One of them transforms formulas into clausules. Later they are grouped to form the edges of the graph. Then the graph is reduced by absorption of clausules. Next module includes the unification algorithm for the formulas with identity. Another module chooses various strategies of

proving theorems. Next module contains adopting procedures of division and simplification of the diagram of the graph. By means of this module it is possible to discover the loops caused by frequent usage of the same lemma. The clausules derived from the considered theorem have priority. Useless edges of the graph resulted from tautologies are reduced. Using this module we often lose completeness of this strategy of proving. All these modules improve effectiveness of this system.

Next system constructed by T. C. Wang [99] is based on resolution. Additionally in each constructed clausule there is information about "history" of clausules. It makes possible to limit the form of the generated proofs. This method finds some special cases of absorption.

In this system there exists a semantical approach to proving theorems. The system will consider only these clausules which are accepted in the model. In [99] we can find examples of proved theorems.

The proving theorem system constructed by S. Wolfram and Ch. Cole and described by A. Bundy in [14] is an interactive system which facilitates manipulation of mathematical expressions. This system can perform the following operations:

1. Decomposition of mathematical expressions,
2. Operations on polynomials,
3. Solution of linear and not linear equations of several variables,
4. Differentation and integration of the wide class of expressions,
5. Operations on matrices,
6. Operations on finite and infinite series (limitation, addition, multiplication).

This system enables the user access to various mathematical environment:

- numerical calculations,
- graphic representation of mathematical expressions,
- advanced programming language,
- interactive communication.

W. Bledsone and M. Tyson [11] constructed Gentzen's interactive system for proving theorems of the first-order predicate calculus. The key idea of the system is based on dividing the problem into many subproblems. It is possible to use mathematical induction. The user can influence the process of searching the proof and indicate the optional rule of conclusion. This system was described by A. Bundy in [14].

Some aspects on automatic theorem proving were described by A. Biela and M. Wojtylak in [8].

A. Trybulec [97] developed the well-known MIZAR proof-checking system based on the resolution method.

11th International Conference, TPHOLs'98, Canberra, Australia, September 27—October 1 was dedicated to current aspects of theorem proving in higher order logics and formal verification and program analysis. Besides the HOL system, the theorem provers Coq, Isabelle, Lambda, Lego, Nuprl and PVS were discussed and published in Proceedings [36].

J. Harrison in [40] combines traditional lines of research in theorem proving and shows the usefulness of real numbers in verification.

This analysis of literature on automatic theorem proving points out that there are many interesting systems.

Our considerations strongly vary from the above-mentioned studies, since our logic contains a built-in notion of program and because these considerations enable us to prove theorems which include programs. Our constructed system enables us to find assumptions which are necessary for the proof of expressions which are not theorems. Then this system looks for the special assumptions during the execution of program and tries to finish the proof. After finishing the proof this system shows us all the adopted assumptions. Using this system the partial correctness and equivalence of programs can be determined.

PART I

Chapter 2

Basic definitions

2.1 The language of AL

To construct a language of algorithmic logic we have to distinguish a set of signs called the alphabet and to give some *syntax rules* of creating syntactically admissible expressions in the language.

The *alphabet* L of algorithmic logic AL consists of the union of disjoint and at most denumerable sets:

1. V the infinite set of *individual variables*,
2. V_0 the infinite set of *propositional variables*, we assume that the set $V_0 \cup V$ is linearly ordered by a certain ordering relation,
3. $\mathcal{N}$ the set of *non-negative integers* and $N = \mathcal{N} \setminus \{0\}$,
4. $\bigcup_{m \in N} P_m$, where P_m is the set of *m-argument predicates*,
5. $\bigcup_{m \in \mathcal{N}} \Phi_m$, where Φ_m denotes the set of *m-argument function symbols*,
6. $\{TRUE, FALSE\}$ the set of *logical constants*,
7. $\{\neg, \wedge, \vee, \rightarrow\}$ the set of *logical connectives*: $\neg$ (negation), $\wedge$ (conjunction), $\vee$ (disjunction) and $\rightarrow$ (implication),
8. $\{\forall\}$ the set of *general quantifier* /$\exists$ means $\neg \forall \neg$/,
9. $\{\bigcup, \bigcap\}$ the set of *existential iteration quantifier* and the *universal iteration quantifier* respectively,
10. {**begin** - ; - **end**, **if** - **then** - **else** - , **while** - **do** -} the set of *program connectives* called *composition, branching* and *iteration* respectively,
11. {(,),/,[,]} the set of *auxiliary signs*. □

The standard definitions of the sets T_o, F_o, S_o, S F of *classical terms, classical open formulas, substitutions* as *assignment instructions, programs,* and *generalized formulas* sometimes called formulas may be found in L. Banachowski [1], G. Mirkowska and A. Salwicki [64] and A. Biela [5]. We recall these definitions.

By the set T_o all *classical terms* we shall understand the least set of expressions closed under the following two formation rules:

t1. If $x \in V$ then $x \in T_o$,
t2. If $\varphi \in \Phi_m$ for some $m \in \mathcal{N}$ and $\tau_1, \ldots, \tau_m \in T_o$ are classical terms then $\varphi(\tau_1, \ldots, \tau_m) \in T_o$. □

By the set F_o of all *classical open formulas* we shall understand the least set of expressions closed under the formation rules:

f1. $V_o \cup \{TRUE, FALSE\} \subset F_o$,
f2. If $\rho \in P_m$ for some $m \in N$ and $\tau_1, \ldots, \tau_m \in T_o$ then $\rho(\tau_1, \ldots, \tau_m) \in F_o$,
f3. If $\alpha, \beta \in F_o$ then $\neg(\alpha)$, $(\alpha \wedge \beta)$, $(\alpha \vee \beta)$, $(\alpha \rightarrow \beta) \in F_o$. □

The set (denote it by At and call *atomic formulas*) is created by usage of f1, f2 formation rules only. By an *elementary formula* we shall understand any classical open formula of the form $\rho(\tau_1, \ldots, \tau_m)$. Let E be the set of all elementary formulas.

The set S_o of *assignment instructions* is the set of all expressions of the form:

(a) $[x_1/\tau_1, \ldots, x_n/\tau_n, a_1/\alpha_1, \ldots, a_m/\alpha_m]$ for $n, m \in N$,
where $x_1, \ldots, x_n$ (respectively $a_1, \ldots, a_m$) are pairwise different individual (respectively propositional) variables, $\tau_1, \ldots, \tau_n$ are classical terms and $\alpha_1, \ldots, \alpha_m$ are classical open formulas. □

The set S of *programs* is the least set containing all elements of S_o closed under the formation rule:

s1. If $\alpha \in F_o$ and $K, M \in S$ then **begin** K; M **end**, **if** α **then** K **else** M, **while** α **do** K $\in S$. □

Sometimes the programs **begin** K; M **end**, **if** α **then** K **else** M, **while** α **do** K will be denoted by $[K\ M]$, $\veebar[\alpha\ K\ M]$, $*[\alpha K]$ respectively. Let us denote $[K_1 [K_2 [\ldots [K_{m-1} K_m] \ldots]]]$ by $[K_1 \ldots K_m]$ for $m \geq 2$

The set T of all generalized terms is the least set containing T_o closed under the following formation rules of construction:
1. If $\tau_1, \ldots, \tau_n$ are generalized terms then $\varphi(\tau_1, \ldots, \tau_n) \in T$,
2. If $K \in S$ and $\tau \in T$ then $K\tau \in T$. □

The set F of *generalized formulas* is the least set satisfying the following conditions:

1. $F_o \subset F$,
2. If $\alpha, \beta \in F$ then $\neg(\alpha)$, $(\alpha \wedge \beta)$, $(\alpha \vee \beta)$, $(\alpha \rightarrow \beta) \in F$,
3. If $K \in S$ and $\alpha \in F$ then $K\alpha$, $\bigcup K\alpha$, $\bigcap K\alpha \in F$, where $\bigcup$ and $\bigcap$ denote the existential and universal iterational quantifiers respectively. □

The set $F_\forall$ of *generalized formulas with quantifiers* is the least set satisfying the following conditions:

q1. $F_o \subset F_\forall$,
q2. If α, $\beta \in F_\forall$ then $\neg(\alpha)$, $(\alpha \wedge \beta)$, $(\alpha \vee \beta)$, $(\alpha \rightarrow \beta) \in F_\forall$,
q3. If $K \in S$ and $\alpha \in F_\forall$ then $K\alpha$, $\bigcup K\alpha$, $\bigcap K\alpha \in F_\forall$, where $\bigcup$ and $\bigcap$ denote the existential and universal iterational quantifiers respectively,
q4. If $\alpha \in F_\forall$ and $x \in V$ then $\exists_x \alpha$, $\forall_x \alpha \in F_\forall$. □

By the *language* of AL we shall mean the system $\mathscr{L} = \langle L, T_o, F_o, S_o, S, F \rangle$ and by the *language with generalized terms* we shall mean the system $\mathscr{L}' = \langle L, T_o, F_o, S_o, S, T, F' \rangle$, where F' additionally is closed under the following formation rule:

(i) If $\tau_1, \ldots, \tau_n$ are generalized terms then $\rho(\tau_1, \ldots, \tau_n) \in F'$, where ρ is an n-argument predicate symbol. □

By the *language of the extended algorithmic logic of the first order, with classical quantifiers* introduced by L. Banachowski [1] we shall mean the system $\mathscr{L}'' = \langle L, T_o, F_o, S_o, S, F_\forall \rangle$. □

We shall denote by $\vartheta(\zeta)$ the set of all individual and propositional variables of the expression ζ. Let $\emptyset$ denotes the empty set and $P(X)$ denotes the set of all subset of the set X.

If $s \in S$ is of the form **(a)** then the expression obtained from the expression ζ by simultaneously replacing all occurrences of the variables x_i, $a_j \in \{x_1, \ldots, x_n, a_1, \ldots, a_m\}$ by the expressions τ_i, α_j for $1 \le i \le n$, $1 \le j \le m$ will be denoted by $s\bar{\zeta}$.

2.2 Realization of an algorithmic language

Let U be a non-empty set and let $\mathscr{B}_o = \langle B_o, \cup, \cap, \mapsto, -, \bigwedge_o, \bigvee_o \rangle$ be a *two-element boolean algebra* with the *unit element* $\bigvee_o$ and the *zero-element* $\bigwedge_o$ and $B_o = \{\bigwedge_o, \bigvee_o\}$, where $-$ is a *complementation* and $\cup$, $\cap$, $\mapsto$ are *binary operations* on B_o such that $x \cap y$ is the *infimum*, $x \cup y$ is the *supremum* and $x \mapsto y = -x \cup y$. Let $U^n = \underbrace{U \times \ldots \times U}_{n-\text{times}}$ be a *Cartesian product* of the *set* U.

By a *valuation* v in the set U and the algebra $\mathscr{B}_o$ we shall understand any mapping of the set of individual variables and propositional variables into U or B_o respectively. The set of all valuations will be denoted by W. □

By a *realization* (see L. Banachowski [1], G. Mirkowska and A. Salwicki [64], A. Biela [5]) of the language $\mathscr{L}'$ in a non-empty set U and in the boolean algebra $\mathscr{B}_o$ we shall understand any mapping $\mathscr{R}$ assigning to each functor φ, a function $\varphi_{\mathscr{R}}: U^n \to U$ and to each predicate ρ, a function $\rho_{\mathscr{R}}: U^n - B_o$. Any realization $\mathscr{R}$ induces mappings $\tau_{\mathscr{R}}: W \cup \{LOOP\} \to U \cup \{LOOP\}$ for $\tau \in T$, $s_{\mathscr{R}}: W \cup \{LOOP\} \to W \cup \{LOOP\}$ for $s \in S_o$, $\alpha_{\mathscr{R}}: W \cup \{LOOP\} \to B_o$ for $\alpha \in F'$ and mappings $K_{\mathscr{R}} \subset W \cup \{LOOP\} \times W \cup \{LOOP\}$ for $K \in S$. $LOOP$ differs from any other element of AL and intuitively means that the value of a program in a realization and a valuation is not defined. We give the precise definitions of these functions.

Let $v \in W$ then

(i) $w_{\mathscr{R}}(v) = v(w)$, $x_{\mathscr{R}}(LOOP) = LOOP$ and $p_{\mathscr{R}}(LOOP) = \bigwedge_o$ for every individual and propositional variable w and for every individual variable x and for every propositional variable p,

(ii) $$\varphi(\tau_1, ..., \tau_n)_{\mathscr{R}}(v) = \begin{cases} \varphi_{\mathscr{R}}(\tau_{1\mathscr{R}}(v), ..., \tau_{n\mathscr{R}}(v)) & \text{if } \tau_{i\mathscr{R}}(v) \text{ are defined} \\ & \text{for every } 1 \leq i \leq n \\ LOOP & \text{in the opposite case} \end{cases}$$

(iii) $$\rho(\tau_1, ..., \tau_n)_{\mathscr{R}}(v) = \begin{cases} \rho_{\mathscr{R}}(\tau_{1\mathscr{R}}(v), ..., \tau_{n\mathscr{R}}(v)) & \text{if } \tau_{i\mathscr{R}}(v) \text{ are defined} \\ & \text{for every } 1 \leq i \leq n \\ \bigwedge_o & \text{in the opposite case} \end{cases}$$

(iv) $s_{\mathscr{R}}(v) = v'$ and $s_{\mathscr{R}}(LOOP) = LOOP$ for every assignment instruction s of the form (a), where

$$v'(z) = \begin{cases} v(z) & \text{for } z \notin \{x_1, ..., x_n, a_1, ..., a_m\} \\ \tau_{i\mathscr{R}}(v) & \text{if } z = x_i \text{ for some } 1 \leq i \leq n \\ \alpha_{j\mathscr{R}}(v) & \text{if } z = a_j \text{ for some } 1 \leq j \leq m \end{cases}$$

Obviously $[\,]_{\mathscr{R}}(v) = v$,

(v) $TRUE_{\mathscr{R}}(v) = \bigvee_o$, $FALSE_{\mathscr{R}}(v) = \bigwedge_o$,

(vi) $(\neg \alpha_{\mathscr{R}})(v) = -\alpha_{\mathscr{R}}(v)$,
$(\alpha \wedge \beta)_{\mathscr{R}}(v) = \alpha_{\mathscr{R}}(v) \cap \beta_{\mathscr{R}}(v)$,
$(\alpha \vee \beta)_{\mathscr{R}}(v) = \alpha_{\mathscr{R}}(v) \cup \beta_{\mathscr{R}}(v)$,
$(\alpha \to \beta)_{\mathscr{R}}(v) = \alpha_{\mathscr{R}}(v) \mapsto \beta_{\mathscr{R}}(v)$, for every classical open formulas $\alpha, \beta \in F_o$,

(vii) If $\alpha \in F_o$ and K, $M \in S$ then

$$[KM]_{\mathfrak{A}}(v) = \begin{cases} M_{\mathfrak{A}}(K_{\mathfrak{A}}(v)) & \text{if } K_{\mathfrak{A}}(v) \text{ and } M_{\mathfrak{A}}(K_{\mathfrak{A}}(v)) \text{ are defined} \\ LOOP & \text{in the opposite case} \end{cases}$$

$$\veebar[\alpha KM]_{\mathfrak{A}}(v) = \begin{cases} K_{\mathfrak{A}}(v) & \text{if } \alpha_{\mathfrak{A}}(v) = \bigvee_0 \text{ and } K_{\mathfrak{A}}(v) \text{ is defined} \\ M_{\mathfrak{A}}(v) & \text{if } \alpha_{\mathfrak{A}}(v) = \bigwedge_0 \text{ and } M_{\mathfrak{A}}(v) \text{ is defined} \\ LOOP & \text{in the opposite case} \end{cases}$$

$$*[\alpha K]_{\mathfrak{A}}(v) = \begin{cases} K^i_{\mathfrak{A}}(v) & \text{where } i \text{ is the natural number such that} \\ & (K^i\alpha)_{\mathfrak{A}}(v) = \bigwedge_0,\ K^i_{\mathfrak{A}}(v) \text{ is defined and} \\ & (K^j\alpha)_{\mathfrak{A}}(v) = \bigvee_0 \text{ for every } j < i \\ LOOP & \text{if such } i \text{ does not exist} \end{cases}$$

(viii) If $K \in S$ and $\tau \in T$ then

$$(K\tau)_{\mathfrak{A}}(v) = \begin{cases} \tau_{\mathfrak{A}}(K_{\mathfrak{A}}(v)) & \text{if } K_{\mathfrak{A}}(v) \text{ is defined} \\ LOOP & \text{otherwise} \end{cases}$$

(ix) If $\alpha \in F'$ and $K \in S$ then

$$(K\alpha)_{\mathfrak{A}}(v) = \begin{cases} \alpha_{\mathfrak{A}}(K_{\mathfrak{A}}(v)) & \text{if } K_{\mathfrak{A}}(v) \text{ is defined} \\ \bigwedge_0 & \text{otherwise} \end{cases}$$

$$(\bigcup K\alpha)_{\mathfrak{A}}(v) = \sup\{(K^i\alpha)_{\mathfrak{A}}(v) : i \in \mathcal{N}\},$$

$$(\bigcap K\alpha)_{\mathfrak{A}}(v) = \inf\{(K^i\alpha)_{\mathfrak{A}}(v) : i \in \mathcal{N}\},$$

where $K^0\alpha = \alpha$ and $K^{i+1}\alpha = K(K^i\alpha)$ for every $\alpha, \beta \in F'$.

(x) The equalities from (vi) for every $\alpha, \beta \in F'$.

If we consider the language $\mathcal{L}''$ then we omit the point (viii) and we change F' into F_V and we add a new point:

(xi) If $\alpha \in F_V$ and $x \in V$ then $(\exists_x \alpha)_{\mathfrak{A}}(v) = \sup\{\alpha_{\mathfrak{A}}(v_j) : j \in U\}$ and $(\forall_x \alpha)_{\mathfrak{A}}(v) = \inf \{\alpha_{\mathfrak{A}}(v_j) : j \in U\}$, where $v_j(x) = j$ and $v_j(z) = v(z)$ for any $v \in W$ and $z \neq x$. □

To illustrate the meaning of the generalized formula of the form Ka for $a \in V_0$ let us consider the language of arithmetic system in the set of nonnegative integers using s as successor function, $\mathbf{0}$ as the number zero and $=$ as the identity relation. For the sequel considerations we shall use alpha $\alpha \equiv \beta$ instead of $(\alpha \to \beta) \wedge (\beta \to \alpha)$. It is easy to see that for K of the form:

$$\veebar[(x = y)[a/FALSE][[u/\mathbf{0}];\ [*[\neg((u = y) \vee (u = x))[u/s(u)]];$$
$$\veebar[(u = x)[a/TRUE][a/FALSE]]]]]$$

the relation $<$ *less than* is definable by using the generalized formula $K\alpha$ in the following way: $x < y \equiv K\alpha$. In other words we can say that in the language with the separate symbol $<$ the above mentioned formula may be an axiom for the relation less than in the arithmetic system. We illustrate the generalized formulas of the form $\bigcup K\alpha$ and $\bigcap K\alpha$:

$$[x/0]\bigcup[x/x+1](x=y), \qquad [x/1]\bigcap[x/x+1]\neg(x=0).$$

We shall say that a generalized formula $\alpha \in F'$ *is valid in the model* $\mathcal{M} = <U, \mathcal{B}_o, \mathcal{R}>$ $/\mathcal{M} \models \alpha/$ iff $\alpha_{\mathcal{R}}(v) = \bigvee_o$ for every $v \in W$. □

We shall say that v is the *valuation in a model* if it is the valuation in the set U and the algebra $\mathcal{B}_o$ of this model. □

A mapping C defined on the set of all subsets of formulas is a *consequence operation* if for every sets X, Y of formulas the following conditions hold:

1. $X \subset C(X)$,
2. $C(C(X)) \subset C(X)$,
3. $C(X) \subset C(Y)$ whenever $X \subset Y$. □

We define the semantic consequence operation $C^{\models}$ in the language $\mathcal{L}'$.

A generalized formula $\alpha \in F'$ is said to be a *semantic consequence operation* of the set $X \subset F'$ (in symbols $\alpha \in C^{\models}(X)$) iff for every model $<U, \mathcal{B}_o, \mathcal{R}>$ the following condition holds: if every generalized formula $\beta \in X$ is valid in the model $<U, \mathcal{B}_o, \mathcal{R}>$, then α is valid in the same model. □

A generalized formula $\alpha \in F'$ is called a *tautology* iff $\alpha \in C^{\models}(\emptyset)$. □

Obviously for the other languages the definitions are analogous. G. Mirkowska [58] proved that the semantic consequence operation is not finitistic so there exist a generalized formula $\alpha \in F'$ and $X \subset F'$ such $\alpha \in C^{\models}(X)$ but $\alpha \notin C^{\models}(X_o)$ for any finite subset $X_o \subset X$. Therefore any axiomatization of $C^{\models}(\emptyset)$ needs at least one rule with an infinite set of premises.

By a *rule* we mean the set of sequents of the form $<X, \alpha>$, where X is a set of formulas called premises and α is a formula called conclusion. □

For any subset D of the set of programs we say that the rule r is *D-admissible* rule of the consequence operation C iff for every sequent $<X, \alpha> \in r$ and for every $K \in D$:

$$\text{if } KX \subset C(\emptyset) \text{ then } K\alpha \in C(\emptyset) \ \square.$$

If D is the set of all primitive programs (i.e. assignment instructions) then instead of saying that the rule r is D-admissible we say that the rule r is an *admissible rule*. □

A rule r is said to be a *derivable rule* of the consequence operation C iff for every sequent $<X, \alpha> \in r$ we get $\alpha \in C(X)$. □

We shall say that a generalized formula α is an element of the set $\mathcal{J}$ iff there exists a classical formula β (i.e. the formula without programs) such that α and β are eqivalent.

A rule r is *finitary* iff for every sequent $<X, \alpha> \in r$ the set X is finite and $X \cup \{\alpha\} \subset \mathcal{J}$. □

We say that a consequence operation C is *complete* iff $C(\alpha) = F$ for every $\alpha \notin C(\emptyset)$. □

2.3 A deductive system for AL

For $\alpha, \beta, \lambda \in F$, $\delta \in F_o$, $s \in S_o$ and $K, M \in S$ we define the notion of an *axiom of algorithmic logic* which will be understood as any generalized formula of one of the following forms:

A1 $(\alpha \to \beta) \to ((\beta \to \lambda) \to (\alpha \to \lambda))$
A2 $\alpha \to (\alpha \vee \beta)$
A3 $\beta \to (\alpha \vee \beta)$
A4 $(\alpha \to \lambda) \to ((\beta \to \lambda) \to ((\alpha \vee \beta) \to \lambda))$
A5 $(\alpha \wedge \beta) \to \beta$
A6 $(\alpha \wedge \beta) \to \alpha$
A7 $(\alpha \to \beta) \to ((\alpha \to \lambda) \to (\alpha \to (\beta \wedge \lambda)))$
A8 $(\alpha \to (\beta \to \lambda)) \to ((\alpha \wedge \beta) \to \lambda)$
A9 $((\alpha \wedge \beta) \to \lambda) \to (\alpha \to (\beta \to \lambda))$
A10 $(\alpha \wedge \neg\alpha) \to \beta$
A11 $(\alpha \to (\alpha \wedge \neg\alpha)) \to \neg\alpha$
A12 $\alpha \vee \neg\alpha$
A13 $TRUE \wedge \neg FALSE$
A14 $s\delta \equiv \overline{s\delta}$
A15 $K(\alpha \vee \beta) \equiv (K\alpha \vee K\beta)$
A16 $K(\alpha \wedge \beta) \equiv (K\alpha \wedge K\beta)$
A17 $K\neg\alpha \to \neg K\alpha$
A18 $K\,TRUE \to (\neg K\alpha \to K\neg\alpha)$
A19 $K(\alpha \to \beta) \to (K\alpha \to \beta)$

A20 $K\,TRUE \to ((K\alpha \to K\beta) \to K(\alpha \to \beta))$
A21 $M \bigcup K\alpha \equiv (M\alpha \vee M \bigcup K(K\alpha))$
A22 $M \bigcap K\alpha \equiv (M\alpha \wedge M \bigcap K(K\alpha))$
A23 $[K\,M]\alpha \equiv K(M\alpha)$
A24 $\veebar[\delta\, K\, M]\alpha \equiv ((\delta \wedge K\alpha) \vee (\neg\delta \wedge M\alpha))$
A25 $*[\delta\, K]\alpha \equiv \bigcup \veebar[\delta\, K[\,]](\neg\delta \wedge \alpha)$
A26 $[\,]\alpha \equiv \alpha$

Let Ax denotes the set of all axioms and let R be the set of rules of inference:

$$r_0 : \frac{\alpha, \alpha \to \beta}{\beta} \qquad r_1 : \frac{\alpha, K\,TRUE}{K\alpha}$$

$$r_2 : \frac{\{\lambda \to M\,K^i\alpha : i \in \mathcal{N}\}}{\lambda \to M \bigcap K\alpha} \qquad r_3 : \frac{\{M\,K^i\alpha \to \lambda : i \in \mathcal{N}\}}{M \bigcup K\alpha \to \lambda}$$

Since two rules have infinite sets of premises, so we define the consequence operation by using ordinal numbers.

Definition 1. *Let γ, μ be any ordinal numbers less than the smallest uncountable ordinal number Ω. The consequence operation of algorithmic logic is defined for $X \subset F$ as follows:*

(1) $C_R^0(X) = Ax \cup X$,
(2) $C_R^{\gamma+1}(X) = C_R^\gamma(X) \cup \{\alpha \in F : <Z, \alpha> \in r$ *for some* $r \in R$ *and* $Z \subset C_R^\gamma(X)\}$,
(3) $C_R^\gamma(X) = \bigcup \{C_R^\mu(X) : \mu < \gamma\}$, *when is a limit ordinal*,
(4) $C_R(X) = \bigcup \{C_R^\gamma(X) : \gamma < \Omega\}$. □

The following theorem was proved by G. Mirkowska [58].

Theorem 1. $C^{\models}(X) = C_R(X)$ *for every* $X \subset F$. □

We shall write $X \vdash \alpha$ instead of $\alpha \in C_R(X)$ *and* $\vdash \alpha$ when X is empty. Any two generalized formulas α and β are *equivalent* iff $\vdash \alpha \equiv \beta$.

Let $\mathscr{A} = <A, f_1, ..., f_n>$ and $\mathscr{B} = <B, g_1, ..., g_n>$ be two similar algebras, i.e. algebras of the same type. A mapping $h : A \to B$ such that $h(f_i(a_1, ..., a_k)) = g_i(h(a_1), ..., h(a_k))$, for all $i \leq n$ and $a_1, ..., a_k \in A$ is called a *homomorphism*.

A homomorphism h is an *endomorphism* if $\mathscr{B} = \mathscr{A}$. Any propositional language can be treated as a special algebra $\mathscr{C} = <C, F_1, ..., F_n>$, where F_i $(1 \leq i \leq n)$ denotes the operator of forming F_i-propositions. Such algebraic treatment of propositional connectives appeared to be very useful.

The rule r_1, as it can be easily seen, reminds of the substitution rule, but really it is not the substitution rule, since a substitution rule ought to be defined as a function i.e. an endomorphism defined on set of atomic formulas with values in the set of all formulas. Unfortunately we can see that this rule transforms any formula $\rho(\tau_1, ..., \tau_n)$ only into the formula of the form $\rho(\tau'_1, ..., \tau'_n)$ but not for example into the conjunction of two formulas. Our aim, however, is to get, maybe under certain restrictions, a standard definition of the rule of substitution.

Chapter 3

The substitution rule

3.1 The notion of (e, g)-function and K_g^e program

In this chapter we shall introduce the notion of program-substitution in AL and we shall prove that it is in accordance with the basic intuitional notion of the standard substitution.

In this paragraph we shall try to separate from all endomorphisms such of them that preserve the main properties of programs.

Let $e : At \to F_o$ be a mapping such that $e(TRUE) = TRUE$ and $e(FALSE) = FALSE$. Let $h^e : F_o \to F_o$ be an extension of e fulfilling the following conditions:

1. $h^e(\alpha) = e(\alpha)$ for $\alpha \in At$,
2. $h^e(\neg\beta) = \neg h^e(\beta)$,
3. $h^e(\beta \bullet \lambda) = h^e(\beta) \bullet h^e(\lambda)$ for $\beta, \lambda \in F_o$ and $\bullet \in \{\wedge, \vee, \to\}$.

It is easily seen that $h^e : F_o \to F_o$ is an endomorphism and that it is the only extension of $e : At \to F_o$ fulfilling (1), (2), and (3).

From all endomorphisms h^e defined on F_o we shall try separating the ones, whose special extensions p to the set F, which will be called the program-substitution, satisfy the following meta-condition:

(b) $p(C_R(\emptyset)) \subset C_R(\emptyset)$.

The condition **(b)** guarantees that the set of all algorithmic theses of AL will be closed under these functions. Now we present two examples showing the difficulties which we will have to overcome.

Example 1. Observe that it is impossible to define $p : F \to F$ for the generalized formula of the form $K\alpha$ by the equality $p(K\alpha) = K(p(\alpha))$, simultaneously maintaining the properties of homomorphism. To visualize this, assume that $e(\rho(x)) = \rho(x) \wedge \rho(y)$ and let x, y, z denote different individual variables. As an immediate consequence we get

$$p([y/z]\,\rho(x)) = [y/z](\rho(x) \wedge \rho(y)) \text{ and } p(\overline{[y/z]\,\rho(x)}) = \rho(x) \wedge \rho(y)$$

since $p(\rho(x)) = h^e(\rho(x)) = e(\rho(x))$. According to the axiom **A14** and (b) we get $\vdash (\rho(x) \wedge \rho(y)) \equiv (\rho(x) \wedge \rho(z))$ which is impossible. ■

Example 2. It can be easily seen that an endomorphism h^e on F_o cannot be extended to the function $p : F \to F$ if p satisfies the condition $p(K\alpha) = K'p(\alpha)$ where $K\alpha$ is the generalized formula and K' is a program.

For this purpose let

$$e(a) = a \wedge FALSE \text{ for } a \in V_o.$$

Then

$$p([a/(a \to a)]\,a) = [a/a \to a]'(a \wedge FALSE)$$

and

$$p(\overline{[a/(a \to a)]\,a}) = p(a) \to p(a).$$

Hence, by (b) and A14 it follows that $\vdash FALSE \equiv TRUE$, which is false. ■

At first we shall introduce a few definitions to illustrate the aim we set at the beginning of the point (b).

Definition 2. *For the further considerations we shall use the symbol g, sometimes with indices, for any one-one mapping of the set $V \cup V_o$ into $V \cup V_o$ such that $g(V) \subset V$ and $g(V_o) \subset V_o$. We denote by G the set of all of these mappings. Any such mappings can be extended to the function g' defined on $T_o \cup F_o$ by putting:*

1. $g'(z) = g(z)$ *for every* $z \in V \cup V_o$,
2. $g'(TRUE) = TRUE$ *and* $g'(FALSE) = FALSE$,
3. $g'(\varphi(\tau_1, \ldots, \tau_n)) = \varphi(g'(\tau_1), \ldots, g'(\tau_n))$ *for any* $\varphi \in \Phi_n$, $n \in N$ *and for any* $\tau_1, \ldots, \tau_n \in T_o$,
4. $g'(\varphi) = \varphi$ *for any* $\varphi \in \Phi_o$,
5. $g'(\rho(\tau_1, \ldots, \tau_n)) = \rho(g'(\tau_1), \ldots, g'(\tau_n))$ *for any* $\rho(\tau_1, \ldots, \tau_n) \in E$, $n \in N$,
6. $g'(\alpha \bullet \beta) = g'(\alpha) \bullet g'(\beta)$ *for* $\bullet \in \{\wedge, \vee, \to\}$ *and* $g'(\neg\alpha) = \neg g'(\alpha)$. □

If s is of the form (a) and f is a mapping from T_o into T_o and from F_o into F_o such that $f(V) \subset V$, $f(V_o) \subset V_o$ and if f is one-one on $V \cup V_o$ then by $f(s)$ we denote the assignment instruction obtained from s by exchanging all expressions of the form x_i, τ_i, a_j, α_j for $f(x_i)$, $f(\tau_i)$, $f(a_j)$, $f(\alpha_j)$, where $1 \leq i \leq n$ and $1 \leq j \leq m$ respectively. Obviously if $s = []$, then $f(s) = []$.

We can notice that the function g' allows us to change the variables inside any classical term and any classical formula. Now we consider an example to explain the connection between the mapping g' defined on $T_o \cup F_o$ and a certain endomorphism.

Example 3. Let $g \in G$ be a mapping such that $g(V \cup V_o) \subset (V \cup V_o) \setminus \vartheta(\alpha)$ for some $\alpha \in F_o$ and let $e: At \to F_o$ be defined in the following way:

$$e(a) = g(a) \wedge \alpha,$$
$$e(TRUE) = TRUE \text{ and } e(FALSE) = FALSE,$$
$$e(\rho(\tau_1, \ldots, \tau_n)) = \rho(g'(\tau_1), \ldots, g'(\tau_n)) \wedge \alpha$$

for every $a \in V_o$, $\rho \in P_n$, $n \in N$ and $\tau_1, \ldots, \tau_n \in T_o$.

Since $g'(\overline{s\tau}) = \overline{g'(s)g'(\tau)}$ for every $\tau \in T_o$ and every $s \in S_o$ and moreover $\overline{g'(s)\alpha} = \alpha$, we get

$$e(\overline{s\rho(\tau_1, \ldots, \tau_n)}) = e(\rho(\overline{s\tau_1}, \ldots, \overline{s\tau_n}))$$
$$= \rho(g'(\overline{s\tau_1}), \ldots, g'(\overline{s\tau_n})) \wedge \alpha = \rho(\overline{g'(s)g'(\tau_1)}, \ldots, \overline{g'(s)g'(\tau_n)}) \wedge \overline{g'(s)\alpha}$$
$$= \overline{g'(s)(\rho(g'(\tau_1), \ldots, g'(\tau_n)) \wedge \alpha)} = \overline{g'(s)e(\rho(\tau_1, \ldots, \tau_n))}.$$

By A14 we get

$$\vdash \overline{g'(s)e(\rho(\tau_1, \ldots, \tau_n))} \equiv g'(s)e(\rho(\tau_1, \ldots, \tau_n)).$$

Thus

$$\vdash e(\overline{s\rho(\tau_1, \ldots, \tau_n)}) \equiv g'(s)e(\rho(\tau_1, \ldots, \tau_n)). \blacksquare$$

Examples 1 and 2 show that the definition of program-substitution on the axiom A14 ought to be very sophisticated. Example 3 shows a way how to do it. Moreover Example 1 shows that if $\vartheta(e(\alpha)) \setminus \vartheta(\alpha) \neq \emptyset$ then we have difficulties with fulfilling the axiom A14 and we overcome them here by using the function g', which enables us to separate variables and which fulfills the equality

$$e(\overline{s\rho(\tau_1, \ldots, \tau_n)}) = \overline{g'(s)e(\rho(\tau_1, \ldots, \tau_n))}.$$

For further considerations we assume that $g \in G$ and g' is the extension of g from Definition 2.

Definition 3. *Let* $g \in G$.

$e \in \mathscr{E}_g$ *iff* (1) $e: At \to F_o$,
(2) $e(TRUE) = TRUE$ *and* $e(FALSE) = FALSE$,
(3) $e(\overline{s\rho(\tau_1, \ldots, \tau_n)}) = \overline{g'(s)e(\rho(\tau_1, \ldots, \tau_n))}$
for any elementary formula $\rho(\tau_1, \ldots, \tau_n)$. $\square$

It is easy to observe that for such a mapping that $e \in \mathscr{E}_g$, $e: At \to F_o$ there exists an endomorphism $h^e: F_o \to F_o$.

Lemma 1. *For every $g \in G$ and for every $e \in \mathscr{E}_g$ we get*

(i) $g(V_o) \cap \vartheta(e(E)) = \emptyset$,
(ii) $g(V \setminus \vartheta(\alpha)) \cap \vartheta(h^e(\alpha)) = \emptyset$ *for every* $\alpha \in F_o$, *such that* $\vartheta(\alpha) \cap V_o = \emptyset$. □

Proof. (i) Assume to the contrary that there exist $a \in V_o$ and $\rho(\tau_1, ..., \tau_n) \in E$ such that $g(a) \in \vartheta(e(\rho(\tau_1, ..., \tau_n)))$. By virtue of Definition 3 we get

$$e(\overline{[a/b]\rho(\tau_1, ..., \tau_n)}) = \overline{g'([a/b])e(\rho(\tau_1, ..., \tau_n))} \text{ for } b \in V_o \text{ and } a \neq b.$$

Hence

$$e(\rho(\tau_1, ..., \tau_n)) = \overline{[g(a)/g(b)]\,e(\rho(\tau_1, ..., \tau_n))}.$$

Since $g(a) \neq g(b)$ then

$$g(a) \notin \vartheta\ (\overline{[g(a)/g(b)]\,e(\rho(\tau_1, ..., \tau_n))}).$$

Thus $g(a) \notin \vartheta(e(\rho(\tau_1, ..., \tau_n)))$ which is impossible.
(ii) Let $y \in V \setminus \vartheta(\alpha)$. If $\alpha \in \{TRUE, FALSE\}$ then $\vartheta(e(\alpha)) = \emptyset$.
If α is of the form $\rho(\tau_1, ..., \tau_n)$ for $\rho(\tau_1, ..., \tau_n) \in E$ then

$$\overline{[y/z]\,\rho(\tau_1, ..., \tau_n)} = \rho(\tau_1, ..., \tau_n).$$

For $z \neq y$ from Definition 2 we get $g(y) \neq g(z)$ and

$$g(y) \notin \vartheta\ (\overline{[g(y)/g(z)]\,e(\rho(\tau_1, ..., \tau_n))}).$$

Hence and from Definition 3 (3) we get $g(y) \notin \vartheta(e(\rho(\tau_1, ..., \tau_n)))$.
Let us assume inductively that (ii) holds for every subformula of α. If α is of the form $\beta \bullet \lambda$ or $\neg\beta$ for some $\bullet \in \{\wedge, \vee, \to\}$ then by the inductive hypothesis we get $g(y) \notin \vartheta(h^e(\beta)) \cup \vartheta(h^e(\lambda))$ in the first case or $g(y) \notin \vartheta(h^e(\beta))$ in the second case. Since h^e is an endomorphism, $g(y) \notin \vartheta(h^e(\alpha))$. ■

It is easy to see that for every g from Definition 2 there exists a function $e: At \to F_o$ such that $e \in \mathscr{E}_g$. Obviously if $e(TRUE) = TRUE$, $e(FALSE) = FALSE$ and $e(\rho(\tau_1, ..., \tau_n)) = \rho(g'(\tau_1), ..., g'(\tau_n))$ then $e \in \mathscr{E}_g$ for a given g from Definition 2. However, it is not true that for every function $e: At \to F_o$ there exists a function g from Definition 2 such that $e \in \mathscr{E}_g$.

We would like to explain the underlying idea of the definition of a program-substitution $p : F \to F$. Look at Example 2 and consider a generalized formula $\alpha = [a/a \to a]a \equiv \overline{[a/a \to a]a}$. From **(b)** and **A14** the generalized formula $p(\alpha)$ should be a theorem of AL, but Example 2 shows that it depends on the value of $p([a/a \to a]a)$.

The above considerations allow us to define the program-substitution $p : F \to F$ by putting the restriction $p/F_o = h^e$ for some mapping $g \in G$ and some $e \in \mathscr{E}_g$. Now we have to decide how to define $p(\alpha)$ for $\alpha \in F \setminus F_o$.

By using a mapping $g \in G$ and $e \in \mathscr{E}_g$ we put

$$p(\alpha) = [g(a)/e(a)]\,([g(a)/g(a) \to g(a)]\, g(a) \equiv (g(a) \to g(a))).$$

By **(b)** we should get $\vdash p(\alpha)$. By **A14** we get

$$\vdash [g(a)/g(a) \to g(a)]\, g(a) \equiv (g(a) \to g(a))).$$

Since the rule r of the scheme $\frac{\alpha}{s\alpha}$ for any $\alpha \in F$ and $s \in S_o$ is a derivable rule in the consequence operation of AL, we get

$$\vdash [g(a)/e(a)]\,([g(a)/g(a) \to g(a)]\, g(a) \equiv (g(a) \to g(a))).$$

Thus $\vdash p(\alpha)$.

Look at $p(\alpha)$ once more. We introduce and explain some abbreviations which will be defined later. Obviously $\vartheta(\alpha) = \{a\}$, so we put $s^\alpha = [g(a)/e(a)]$ and we changed $K = [a/a \to a]$ for $K^e_g = [g(a)/g(a) \to g(a)]$. Later we shall see that in general if an elementary formula occurs in a program K then K^e_g really depends on a function $e \in \mathscr{E}_g$, and any propositional variable $a \in V_o$ will be changed by $g(a)$. Therefore $p(\alpha) = s^\alpha(K^e_g g(a) \equiv (g(a) \to g(a)))$.

Now let us consider the generalized formula

$$\beta = \overline{[a/\rho'(x), y/z]\, \rho(x)} \equiv [a/\rho'(x), y/z]\, \rho(x)$$

for different individual variables x, y, $\in V$. Let $g \in G$ and $e \in \mathscr{E}_g$. We put

$$p(\beta) = [g(a)/e(a)]\,(e(\rho(x)) \equiv [g(a)/e(\rho'(x)), g(y)/g(z)]\, e(\rho(x))).$$

By Lemma 1

$$\overline{[g(a)/e(\rho'(x)), g(y)/g(z)]\, e(\rho(x))} = e(\rho(x)).$$

Hence and by r_1, **A14** we get $\vdash p(\beta)$.

Look at $p(\beta)$ once more. Since $\vartheta(\beta) \cap V_o = \{a\}$, we put $s^\beta = [g(a)/e(a)]$ and we change

$$K = [a/\rho'(x), y/z] \text{ for } K_g^e = [g(a)/e(\rho'(x)), g(y)/g(z)]$$

and

$$\rho(x) \text{ for } e(\rho(x)).$$

Thus

$$p(\beta) = s^\beta(e(\rho(x)) \equiv K_g^e e(\rho(x))).$$

These examples show us that for any generalized formula $\alpha \in F \setminus F_o$ the notion $p(\beta)$ needs the following expressions: the assignment instruction s^β and the program K_g^e for every $K \in F$. But K_g^e may be defined by using the function which transforms $x \in V$, $a \in V_o$, $\rho(\tau_1, ..., \tau_n) \in E$ into $g(x)$, $g(a)$, $e(\rho(\tau_1, ..., \tau_n))$ respectively.

Definition 4. *Let $g \in G$ and $e \in \mathscr{E}_g$. The function $u: At \to F_o$ is (e, g)-function iff $u(\alpha) = g(\alpha)$ for $\alpha \in V_o$ and $u(\alpha) = e(\alpha)$ for $\alpha \in At \setminus V_o$.* □

If u is (e, g)-function then there exists an endomorphism h^u defined on F_o.

Definition 5. *For any program $K \in S$ and any function $g \in G$ and $e \in \mathscr{E}_g$ if u is (e, g)-function then we define K_g^e as follows:*

1. *If $K = []$, then $K_g^e = []$,*
2. *If K is of the form* (a) *i.e. K is an assignment instruction then $K_g^e = [g(x_1)/g'(\tau_1), ..., g(x_n)/g'(\tau_n), g(a_1)/h^u(\alpha_1), ..., g(a_m)/h^u(\alpha_m)]$,*
3. *If K is of the form $[MN]$, $\veebar[\delta MN]$ or $*[\delta M]$, then K_g^e equals $[M_g^e N_g^e]$, $\veebar[h^u(\delta) M_g^e N_g^e]$ or $*[h^u(\delta) M_g^e]$ respectively.* □

Definition 6. *Let H be an endomorphism on F such that the restriction $H/F_o = h^u$ for some $g \in G$, $e \in \mathscr{E}_g$ and (e, g)-function u. Moreover we assume that for every $\alpha \in F$ and $K \in S$ the function H satisfies the following conditions:*

$$H(K\alpha) = K_g^e H(\alpha),\ H(\bigcup K\alpha) = \bigcup K_g^e H(\alpha),\ H(\bigcap K\alpha) = \bigcap K_g^e H(\alpha). \ \square$$

3.2 Program-substitution

In this paragraph we shall introduce the notion of program-substitution. This definition needs the above defined endomorphism H and a special assignment instruction s^α for every $\alpha \in F$. Now we define the notion of s^α.

For every generalized formula $\alpha \in F$ such that $\vartheta(\alpha) \cap V_o = \{a_1, ..., a_m\}$ and for a couple of functions f, f' such that $f: T_o \cup F_o \to T_o \cup F_o$, f restricted to V_o is a one-one mapping from V_o into V_o, $f': F_o \to F_o$, we introduce the following abbreviation:

$$s^\alpha = [f(a_1)/f'(a_1), ..., f(a_m)/f'(a_m)].$$

If $\vartheta(\alpha) \cap V_o = \emptyset$ then we put $s^\alpha = [\,]$. Further we shall say that s^α *is designated by* $<f, f'>$.

Definition 7. *Let $g \in G$, $e \in \mathscr{E}_g$ and $p: F \to F$. We shall say that a* mapping p is defined by using g and e *iff for* (e, g)*-function u and an endomorphism H defined on F such that $H/F_o = h^u$ the following properties hold:*
(1) *H fulfils all conditions from Definition 6,*

$$p(\alpha) = \begin{cases} h^e(\alpha) & for\ \alpha \in F_o \\ s^\alpha H(\alpha) & for\ \alpha \in F \setminus F_o \end{cases}$$

where s^α is designated by the couple $<g, e>$. □

Definition 8. *Let $p: F \to F$. We shall say that a mapping p is a program-substitution ($p \in Sb$) iff for some $g \in G$ and $e \in \mathscr{E}_g$, p is defined by using g and e.* □

Let us observe that the last condition (3) in Definition 3 is essential in such a meaning that if we define the notion of program-substitution changing only the definition of the set $\mathscr{E}_g$ for $g \in G$ by missing in Definition 3 the point (3) then we can show that for some $g \in G$, and e from $\mathscr{E}_g$ without the point (3), there exists a program-substitution for which the property **(b)** does not hold. Let x, y, z, denote some different individual variables and $g \in G$ such that $g(y) \neq x$. Then for $e: At \to F_o$ such that $e(TRUE) = TRUE$, $e(FALSE) = FALSE$ and $e(\rho(x)) = \rho'(x, g(y))$ we can show that for an axiom α of the form

$$\overline{[y/z]\rho(x)} \equiv [y/z]\rho(x)$$

the following property holds:

$$e\overline{[(y/z]\rho(x))} \neq \overline{g'([y/z])e(\rho(x))}$$

and that the generalized formula $p(\alpha)$ is not a thesis. Hence (3) from Definition 3 and **(b)** are false.

Lemma 2. *If $g \in G$, $e \in \mathscr{E}_g$, $s \in S_o$ and u is (e,g)-function then for any $\alpha \in F_o$ and $\beta \in F$ we get*

(i) $\overline{s_g^e h^u(\alpha)} = h^u(\overline{s\alpha})$,

(ii) $\overline{s^\beta h^u(\alpha)} = h^e(\alpha)$, *for $V_o \cap \vartheta(\alpha) \subset \vartheta(\beta)$, where s^β is designated by the couple $<g, e>$.* □

Proof. (i) Let $s \in S_o$ be of the form (a). If $\alpha \in V_o$ and $\alpha \in \{a_1, ..., a_m\}$ then $\alpha = a_i$ for some $i \in \{1, ..., m\}$. Thus $\overline{s_g^e h^u(\alpha)} =$
$\overline{[g(x_1)/g'(\tau_1), ..., g(x_n)/g'(\tau_n), g(a_1)/h^u(\alpha_1), ..., g(a_m)/h^u(\alpha_m)]\, g(a_i)} = h^u(\alpha_i) = h^u(\overline{s\alpha})$.

Since g is a one-one function, we get $g(\alpha) \notin \{g(x_1), ..., g(x_n), g(a_1), ..., g(a_m)\}$ for $\alpha \notin \{a_1, ..., a_m\}$. Hence $\overline{s_g^e h^u(\alpha)} = g(\alpha) = h^u(\overline{s\alpha})$. Obviously if α is of the form *TRUE* or *FALSE* then (i) holds.

Assume that α is of the form $\rho(\tau_1, ..., \tau_n)$ and $\rho(\tau_1, ..., \tau_n) \in E$. First we shall prove that $\overline{s_g^e \eta} = \overline{g'(s)\eta}$ for every $\eta \in \vartheta(e(\rho(\tau_1, ..., \tau_n)))$.

In the case $\eta \notin \{g(x_1), ..., g(x_n), g(a_1), ..., g(a_m)\}$ we get $\overline{s_g^e \eta} = \eta = \overline{g'(s)\eta}$. If $\eta = g(x_j)$ for some $j \in \{1, ..., n\}$ then $\overline{s_g^e \eta} = g'(\tau_j) = \overline{g'(s)\eta}$. Let us observe that $\eta = g(a_i)$ for some $i \in \{1, ..., m\}$ does not hold for in the opposite case using the assumption we get $\eta \in \vartheta(e(\rho(\tau_1, ..., \tau_n))) \cap V_o$. Hence and by Lemma 1 $\eta \notin g(V_o)$ which is impossible.

Since $\overline{s_g^e \eta} = \overline{g'(s)\eta}$ for every $\eta \in \vartheta(e(\rho(\tau_1, ..., \tau_n)))$, we get

$$\overline{s_g^e e(\rho(\tau_1, ..., \tau_n))} = \overline{g'(s) e(\rho(\tau_1, ..., \tau_n))}$$

Note that $e \in \mathscr{E}_g$, so by Definition 3 we conclude that

$$\overline{s_g^e e(\rho(\tau_1, ..., \tau_n))} = e(\overline{s\rho(\tau_1, ..., \tau_n)}) = u(\overline{s\rho(\tau_1, ..., \tau_n)}).$$

Consequently $\overline{s_g^e h^u(\alpha)} = h^u(\overline{s\alpha})$.

If the theorem holds for $\beta, \lambda \in F_o$ then from the property of endomorphism it also holds for $\alpha \in \{\beta \wedge \lambda, \beta \vee \lambda, \beta \to \lambda, \neg\beta\}$.

(ii) Obviously for $\alpha \in \{TRUE, FALSE\}$ the theorem holds. If $\alpha \in V_o$ then by assumption $\alpha \in \vartheta(\beta)$. As a result $\overline{s^\beta h^u(\alpha)} = \overline{s^\beta g(\alpha)} = e(\alpha)$. Now suppose that α is of the form $\rho(\tau_1, ..., \tau_n)$ for some $\rho(\tau_1, ..., \tau_n) \in E$. By Definition 4 we get

$$\overline{s^\beta h^u(\rho(\tau_1, ..., \tau_n))} = \overline{s^\beta e(\rho(\tau_1, ..., \tau_n))}.$$

Let

$$s^{\beta} = [g(b_1)/e(b_1), \ldots, g(b_l)/e(b_l)].$$

By Lemma 1 (i) we conclude that $\{g(b_1), \ldots, g(b_l)\} \cap \vartheta(\overline{e(\rho(\tau_1, \ldots, \tau_n))}) = \emptyset$. Therefore $\overline{s^{\beta} e(\rho(\tau_1, \ldots, \tau_n))} = e(\rho(\tau_1, \ldots, \tau_n))$. Hence $\overline{s^{\beta} h^u(\alpha)} = h^e(\alpha)$. Since $h^u: F_o \to F_o$ is an endomorphism, by inductive hypothesis for λ, $\delta \in F_o$ we get the equality $\overline{s^{\beta} h^u(\alpha)} = h^e(\alpha)$ for $\alpha \in \{\lambda \wedge \delta, \lambda \vee \delta, \lambda \to \delta, \neg\lambda\}$ such that $V_o \cap \vartheta(\alpha) \subset \vartheta(\beta)$. ■

Now we present an example to explain the effect of program-substitutions.

Example 4. Let $a \in V_o$, x, $y \in V$ and $x \neq y$. It can be easily seen that for the program-substitution $p \in Sb$ defined by using $g \in G$ and $e \in \mathscr{E}_o$ and for the generalized formula $\alpha = a \wedge [x/y]\, \rho(x)$ we get

$$p(\alpha) = s^{\alpha} H(\alpha) = [g(a)/e(a)]\,(H(a) \wedge H([x/y]\, \rho(x)))$$

$$= [g(a)/e(a)]\,(h^u(a) \wedge [x/y]^{*}_{g} H(\rho(x)))$$

$$= [g(a)/e(a)]\,(g(a) \wedge [g(x)/g(y)]\, e(\rho(x)).$$

By A14 we get

$$\vdash [g(x)/g(y)]\, e(\rho(x)) \equiv \overline{[g(x)/g(y)]\, e(\rho(x))}.$$

Since $h^u(\rho(x)) = e(\rho(x))$, Lemma 2 (i) allows us to conclude that

$$\overline{[g(x)/g(y)]\, e(\rho(x))} = e(\rho(y)).$$

Hence

$$\vdash [g(x)/g(y)]\, e(\rho(x)) \equiv e(\rho(x)).$$

Therefore we conclude that

$$\vdash g(a) \wedge [g(x)/g(y)]\, e(\rho(x)) \equiv (g(a) \wedge e(\rho(y))).$$

Since for any β, $\lambda \in F$ and $K \in S$ the rule r' of the scheme $\dfrac{\lambda \equiv \beta}{K\lambda \equiv K\beta}$ is a derivable rule of the consequence operation of AL, we get

$$\vdash [g(a)/e(a)]\,(g(a) \wedge [g(x)/g(y)]\, e(\rho(x))) \equiv [g(a)/e(a)]\,(g(a) \wedge e(\rho(y))).$$

By A14 we get

$$\vdash [g(a)/e(a)]\,(g(a) \wedge e(\rho(y))) \equiv (e(a) \wedge [g(a)/e(a)]\, e(\rho(y)).$$

Since $h^u(a) = g(a)$ and $h^u(\rho(x)) = e(\rho(x))$, we get $\overline{[g(a)/e(a)]\, e(\rho(y))} = e(\rho(y))$ by Lemma 2 (ii). By the above considerations we get $\vdash p(\alpha) \equiv (e(a) \wedge e(\rho(y)))$. Moreover by A14 $\vdash \alpha \equiv (a \wedge \rho(y))$. Therefore we can say intuitively that p transforms a; $\rho(y)$ into $e(a)$; $e(\rho(y))$ respectively. ■

We can find in Chapter 4 in Theorem 9 other examples of program-substitution.

We shall prove that any program-substitution $p \in Sb$ is in accordance with our intuition. Now we consider the condition **(b)**.

Theorem 2. *Algorithmic logic is closed under program-substitutions.* $\square$

Proof. Let $g \in G$, $e \in \mathscr{E}_g$ and let the mapping $p \in Sb$ be defined by using g and e. Moreover let u be (e, g)-function and let $H : F \to F$ be an endomorphism such that the restriction $H/F = h^u$ and

$$p(\alpha) = \begin{cases} h^e(\alpha) & \text{for } \alpha \in F_o \\ s^e H(\alpha) & \text{for } \alpha \in F \setminus F_o. \end{cases}$$

It suffices to prove by induction on the length of the formula α that the following inclusion holds:

$$p(C_R^\gamma(\emptyset)) \subset C_R(\emptyset) \text{ for any ordinal number } \gamma < \Omega.$$

At first we assume that $\gamma = 0$. If $\alpha \in Ax \cap F_o$ then $h^e(\alpha) \in Ax$. Since $p/F_o = h^e$, $\vdash p(\alpha)$. In the case $\alpha \in Ax \setminus F_o$ we get $p(\alpha) = s^e H(\alpha)$. Since $\vdash s^e TRUE$, it suffices to prove that $\vdash H(\alpha)$ using r_1. It can be easily seen that for α being one of the axioms of the form **A1-A13** or **A15-A26** we get $\vdash H(\alpha)$ because H is a homomorphism. Let $\alpha = s\delta \equiv \overline{s\delta}$ for some classical open formula $\delta \in F_o$. By Lemma 2 (i) and by applying **A14** we get

$$\vdash s_g^e h^u(\delta) \equiv h^u(\overline{s\delta}), \text{ so } \vdash H(\alpha).$$

We assume inductively that $p(C_R^\mu(\emptyset)) \subset C_R(\emptyset)$ for every ordinal number μ such that $\mu < \gamma$.

In the first case suppose that $\gamma = \mu_o + 1$ for some ordinal number μ_o and let $\alpha \in p(C_R^\gamma(\emptyset))$. Hence and by Definition 1 we get $\alpha \in p(C_R^{\mu_o}(\emptyset))$ and then by the inductive hypothesis $\vdash p(\alpha)$, or there exist $X \subset C_R^{\mu_o}(\emptyset)$, $\beta \in F$ and $r \in R$ such that $<X, \beta> \in r$ and $\alpha = p(\beta)$. Since $H(Ax) \subset C_R(\emptyset)$ and $<H(Y), H(\lambda)> \in r'$ for every $r' \in R$ and for every $<Y, \lambda> \in r'$, we get $H(C_R(\emptyset)) \subset C_R(\emptyset)$. Thus we get $H(X) \subset C_R(\emptyset)$.

Moreover $<H(X), H(\beta)> \in r$, so $\vdash H(\beta)$. Applying the rule r_1 we conclude that $\vdash s^\beta H(\beta)$ for s^β designated by $<g, e>$. Therefore if $\beta \notin F_o$, then $\vdash p(\beta)$ and simultaneously $\vdash \alpha$, or if $\beta \in F_o$, then $\alpha = h^e(\beta)$ and by **A14** $\vdash \overline{s^\beta H(\beta)}$, which by Lemma 2 (ii) gives $\vdash \alpha$.

In the second case for γ being a limit ordinal number and by the inductive hypothesis we get $\bigcup \{p(C_R^{\mu}(\emptyset)) : \mu < \gamma\} \subset C_R(\emptyset)$.

Hence $p(C_R^{\gamma}(\emptyset)) \subset C_R(\emptyset)$. ■

3.3 Basic properties of program-substitution

The aim of this chapter is to prove that any program-substitution maintains the properties of an endomorphism without being an endomorphism.

Definition 9. *Let us consider the set X of pairs $<\alpha_1, \alpha_2>$, where $\alpha_1, \alpha_2 \in F$ and where α_1 is equal to α_2 or $<\alpha_1, \alpha_2>$ is one of the following forms:*

(1) $<\overline{ss_1 a}, ss_1 a>$ *for* $a \in V_0$,
(2) $<\overline{ss_1 \rho(\tau_1, ..., \tau_n)}, ss_1 \rho(\tau_1, ..., \tau_n)>$ *for* $\rho(\tau_1, ..., \tau_n) \in E$,
(3) $<s\alpha, s \neg\alpha>$,
(4) $<s\alpha, s(\alpha \vee \beta>$,
(5) $<s\beta, s(\alpha \vee \beta)>$,
(6) $<s\alpha, s(\alpha \wedge \beta)>$,
(7) $<s\beta, s(\alpha \wedge \beta)>$,
(8) $<s\alpha, s(\alpha \rightarrow \beta)>$,
(9) $<s\beta, s(\alpha \rightarrow \beta)>$,
(10) $<s(K(M\alpha)), s([K M]\alpha)>$,
(11) $<s(\alpha \wedge K\beta), s \veebar [\alpha K M]\beta)>$,
(12) $<s(\neg\alpha \wedge M\beta), s \veebar [\alpha K M]\beta)>$,
(13) $<s \veebar [\alpha K [\,]]^i (\beta \wedge \neg\alpha), s(*[\alpha K]\beta)>$ *for every* $i \in \mathcal{N}$,
(14) $<s(K^i\alpha), s \bigcup K\alpha>$ *for every* $i \in \mathcal{N}$,
(15) $<s(K^i\alpha), s \bigcap K\alpha>$ *for every* $i \in \mathcal{N}$,

where K, $M \in S$ are programs, α, $\beta \in F$ and where s is either a sequence of assignment instructions $s_1 \ldots s_k$, $k \in N$ or an empty sequence. □

We introduce (cf. G. Mirkowska [59]) the binary relation $<$ in F for any α, $\beta \in F : \alpha < \beta$ iff there exist $\alpha_1, ..., \alpha_n \in F$ such that $\alpha_1 = \alpha$, $\alpha_n = \beta$ and for every $i \in \{1, ..., n-1\}$ the pair $<\alpha_{i}, \alpha_{i+1}>$ is an element of X.

Let us notice that the binary relation $<$ is an ordering on F such that any non-empty subset $Z \subset F$ contains a minimal element.

Now we shall prove that the logical value of the formula α does not depend on the propositional variables which do not belong to the set of propositional variables of the formula α.

Lemma 3. *For any generalized formulas α, $\beta \in F$ and any $g \in G$, $e \in \mathscr{E}_g$, (e, g)-function u and for an endomorphism $H : F \rightarrow F$ fulfilling the conditions* *from Definition 6, the following property holds: if $\vartheta(\alpha) \cap V_o \subset \vartheta(\beta)$, then $\vdash s^\beta H(\alpha) \equiv s^\alpha H(\alpha)$ where s^β and s^α are designated by $<g, e>$.* $\square$

Proof. The proof is by induction on the relation $<$ from Definition 9.

Case 1. If α is a minimal element of the relation $<$ then $\alpha \in F_o$. Hence by Lemma 2 (ii) we get $s^\beta h^u(\alpha) = h^e(\alpha)$ and $s^\alpha h^u(\alpha) = h^e(\alpha)$. Obviously the restriction $H/F = h^u$. Hence and by **A14** the induction basis is proved.

Case 2. Let $\alpha \in F$ and suppose that the thesis holds for every generalized formula $\alpha' \in F$ such that $\alpha' < \alpha$. Moreover assume that $\vartheta(\alpha) \cap V_o \subset \vartheta(\beta)$.

Case 2.1. If $\alpha \in F_o$ then Lemma 2 (ii) ends the proof of this case.

Case 2.2. If α is of the form $s_1 \ldots s_m\delta$ for some $\delta \in F_o$ then we use the abbreviation $\lambda = s_1 \ldots s_{m-1} \overline{s_m\delta}$. Since $H(C_R(\emptyset)) \subset C_R(\emptyset)$, we get $\vdash s^\beta H(\alpha) \equiv s^\beta H(\lambda)$ and $\vdash s^\alpha H(\alpha) \equiv s^\alpha H(\lambda)$ according to the scheme **A14** and by r_1 and r' from Example 4. Since $\lambda < \alpha$, $\vartheta(\lambda) \cap V_o \subset \vartheta(\alpha) \cap \vartheta(\beta)$, we get by the inductive assumption $\vdash s^\alpha H(\lambda) \equiv s^\lambda H(\lambda)$ and $\vdash s^\beta H(\lambda) \equiv s^\lambda H(\lambda)$. As an immediate consequence we get $\vdash s^\beta H(\alpha) \equiv s^\alpha H(\alpha)$.

Case 2.3. Let α be of the form $s_1 \ldots s_m \underline{\vee}[\delta KM]\lambda$, $\alpha_1 = s_1 \ldots s_m(\neg\delta \wedge M\lambda)$ and $\alpha_2 = s_1 \ldots s_m(\delta \wedge K\lambda)$. We can observe that $\alpha_1 < \alpha$, $\alpha_2 < \alpha, \vartheta(\alpha_1) \cap V_o \subset \vartheta(\alpha) \cap \vartheta(\beta)$. Therefore using the inductive argument we get $\vdash s^\alpha H(\alpha_1) \equiv s^{\alpha_1} H(\alpha_1)$ and $\vdash s^\beta H(\alpha_1) \equiv s^{\alpha_1} H(\alpha_1)$. In an analogous way we infer that $\vdash s^\alpha H(\alpha_2) \equiv s^{\alpha_2} H(\alpha_2)$ and $\vdash s^\beta H(\alpha_2) \equiv s^{\alpha_2} H(\alpha_2)$. Hence, by **A24** and r_1 we get $\vdash s^\beta H(\alpha) \equiv s^\alpha H(\alpha_2 \vee \alpha_1)$ which by **A24** proves the thesis for this case.

Case 2.4. If α is of the form $s_1 \ldots s_m [K M] \lambda$ then we use the abbreviation $\alpha_1 = s_1 \ldots s_m KM\lambda$. Similarly by the inductive hypothesis, **A23** and r_1 we obtain the thesis for this case.

Case 2.5. Let us assume that α is of the form $s_1 \ldots s_m \bigcap K\lambda$. We denote

$$\lambda_i = s_1 \ldots s_m K^i \lambda \text{ for any } i \in \mathcal{N}.$$

Obviously $\vartheta(\lambda_i) \cap V_o \subset \vartheta(\beta) \cap \vartheta(\alpha)$. Observe that $\lambda_i < \alpha$ for every $i \in \mathcal{N}$. Consequently, by assumption $\vdash s^\beta H(\lambda_i) \equiv s^{\lambda_i} H(\lambda_i)$ and $\vdash s^\alpha H(\lambda_i) \equiv s^{\lambda_i} H(\lambda_i)$ for every $i \in \mathcal{N}$. Since $\vdash \bigcap K\delta \rightarrow K^i\delta$ for any $\delta \in F$, $i \in \mathcal{N}$ and every $K \in S$, we get $\vdash s^\alpha H(\alpha) \rightarrow s^\alpha H(\lambda_i)$ and $\vdash s^\beta H(\alpha) \rightarrow s^\beta H(\lambda_i)$ for every $i \in \mathcal{N}$ using r_1 and r' from Example 4. Hence and by **A23** for every $i \in \mathcal{N}$ it follows:

$$\vdash s^\alpha H(\alpha) \rightarrow [s^\beta [s_1 \ldots s_m]_g^e](K_g^e)^i H(\lambda)$$

and

$$\vdash s^\beta H(\alpha) \rightarrow [s^\alpha [s_1 \ldots s_m]_g^e](K_g^e)^i H(\lambda).$$

Using r_2 and **A23** we conclude that $\vdash s^\beta H(\alpha) \equiv s^\alpha H(\alpha)$.

Case 2.6. If α is of the form $s_1 \ldots s_m \bigcup K\lambda$ then by analogous considerations the thesis holds for this case, since $\vdash K^i\delta \to \bigcup K\delta$ for any $\delta \in F$, $i \in \mathcal{N}$ and every $K \in S$.

Case 2.7. Let α be of the form $s_1 \ldots s_m * [\delta K]\lambda$. We use the abbreviations $\delta_i = s_1 \ldots s_m (\underline{\vee}[\delta K[\,]]^i(\neg\delta \wedge \lambda)$ for every $i \in \mathcal{N}$. Since $\vartheta(\delta_i) \cap V_o \subset \vartheta(\alpha) \cap \vartheta(\beta)$ and $\delta_i < \alpha$ for every $i \in \mathcal{N}$, we get by the inductive hypothesis $\vdash s^\alpha H(\delta_i) \equiv s^{\delta_i}H(\delta_i)$ and $\vdash s^\beta H(\delta_i) \equiv s^{\delta_i}H(\delta_i)$ for every $i \in \mathcal{N}$.

Hence and by **A23** we get

$$\vdash s^\beta H(\delta_i) \to [s^\alpha [s_1 \ldots s_m]^e_g] (\underline{\vee}(\delta K[\,]]^e_g)^i H(\neg\delta \wedge \lambda)$$

and moreover

$$\vdash s^\alpha H(\delta_i) \to [s^\beta [s_1 \ldots s_m]^e_g] (\underline{\vee}(\delta K[\,]]^e_g)^i H(\quad \delta \wedge \lambda).$$

Clearly $\vdash s_1 \ldots s_m M^i\beta' \to s_1 \ldots s_m \bigcup M\beta'$ for any $\beta' \in F$, $M \in S$ and $i \in \mathcal{N}$. Hence, by **A25** and **A23** it follows that

$$\vdash s^\beta H(\delta_i) \to s^\alpha H(\alpha) \text{ and } \vdash s^\alpha H(\delta_i) \to s^\beta H(\alpha) \text{ for every } i \in \mathcal{N}.$$

Transforming for every $i \in \mathcal{N}$ the antecedents in the above two theses according to the schema **A23** and using the rule r_3 and **A25** we get

$$\vdash s^\beta H(\alpha) \to s^\alpha H(\alpha) \text{ and } \vdash s^\alpha H(\alpha) \to s^\beta H(\alpha),$$

which ends the proof for this case.

Case 2.8. If α is of the form

$$s_1 \ldots s_m(\lambda \wedge \delta),\ s_1 \ldots s_m(\lambda \vee \delta),\ s_1 \ldots s_m(\lambda \to \delta),\ s_1 \ldots s_m(\neg\lambda)$$

then by the inductive hypothesis, the thesis of the above lemma holds. ■

Applying Lemma 3 we get the basic properties of program-substitution.

Theorem 3. *For every program-substitution $p \in Sb$ and for all generalized formulas α, β:*

(i) $\vdash p(\alpha \to \beta) \equiv (p(\alpha) \to p(\beta))$,
(ii) $\vdash p(\alpha \wedge \beta) \equiv (p(\alpha) \wedge p(\beta))$,
(iii) $\vdash p(\alpha \vee \beta) \equiv (p(\alpha) \vee p(\beta))$,
(iv) $\vdash p(\neg\alpha) \equiv \neg p(\alpha)$. □

Proof. We shall prove only the point (i) since the proofs of the other points are analogous. Let us assume that $p \in Sb$ is defined by using some $g \in G$ and $e \in \mathscr{E}_g$. Moreover let

$$p(\alpha) = \begin{cases} h^e(\alpha) & \text{for } \alpha \in F_o \\ s^\alpha H(\alpha) & \text{for } \alpha \in F \setminus F_o \end{cases}$$

where s^α is designated by a couple $<g, e>$ and H fulfills the conditions from Definition 6. If α, $\beta \in F_o$ then (i) holds, since e is an endomorphism on F_o.

Now suppose that $\alpha \notin F_o$ and $\beta \notin F_o$. Thus $p(\alpha \to \beta) = s^{\alpha\to\beta}(H(\alpha) \to H(\beta))$. Since $\vdash s^{\alpha\to\beta}TRUE$, we get

$$\vdash p(\alpha \to \beta) \equiv (s^{\alpha\to\beta}H(\alpha) \to s^{\alpha\to\beta}H(\beta)) \text{ by } \mathbf{A19} \text{ and } \mathbf{A20}.$$

Hence and by Lemma 3 we conclude that $\vdash p(\alpha \to \beta) \equiv (s^\alpha H(\alpha) \to s^\beta H(\beta))$, which ends the proof for this case.

Now we consider the case $\alpha \notin F_o$ and $\beta \in F_o$. Then $p(\alpha) = s^\alpha H(\alpha)$ and $p(\alpha \to \beta) = s^{\alpha\to\beta}H(\alpha \to \beta) = s^{\alpha\to\beta}(H(\alpha) \to H(\beta))$. By **A19** and **A20** we get $\vdash s^{\alpha\to\beta}(H(\alpha) \to H(\beta)) \equiv (s^{\alpha\to\beta}H(\alpha) \to s^{\alpha\to\beta}H(\beta))$. By Lemma 3 $\vdash s^{\alpha\to\beta}H(\alpha) \equiv s^\alpha H(\alpha)$ and $\vdash s^{\alpha\to\beta}H(\beta) \equiv s^\beta H(\beta)$. Hence we conclude that $\vdash p(\alpha \to \beta) \equiv (s^\alpha H(\alpha) \to s^\beta H(\beta))$. Since $\alpha \notin F_o$, $p(\alpha) = s^\alpha H(\alpha)$. Moreover by Lemma 2 (ii) we get $s^\beta H(\beta) = h^e(\beta) = p(\beta)$. By A14 $\vdash s^\beta H(\beta) \equiv \overline{s^\beta H(\beta)}$ so $\vdash s^{\alpha\to\beta}(H(\alpha) \to H(\beta)) \equiv (p(\alpha) \to p(\beta))$. The case $\beta \notin F_o$ and $\alpha \in F_o$ is analogous. ■

Corollary 1. *As an immediate consequence of Theorem 3 and Theorem 2 we conclude that for any generalized formulas* α, $\beta \in F$ *and for every program-substitution* $p: F \to F$, *if* $\vdash \alpha \equiv \beta$ *then* $\vdash p(\alpha) \equiv p(\beta)$. □

The next two subsections introduce the notion of program-substitution in algorithmic logic with generalized terms, quantifiers and with non-deterministic programs. We can omit them while reading the paper for the first time. Therefore these two subsections will be printed in italics.

3.4 Program-substitution in AL with generalized terms

In this chapter we shall to repeat the main results which were proved in earlier paragraphs for the case of generalized formulas. To illustrate the intuitive meaning of the generalized terms of the form Kx *for* $K \in S$ *and* $x \in V$ *let us consider the language of integers with s as succesor and* $\mathbf{0}$, $+$, $-$, $\cdot$, $=$ *as well-known symbols. We consider the generalized formula of the form* $x_1 + x_2 + x_3 = Ky$ *where* K *is of the form*

$$[[y/\mathbf{0}, i/\mathbf{1}] * [i \leq 3\, [[y/y + x_i]\, [i/s(i)]]]].$$

Obviously in the intuitive meaning the term Ky may be understood as a definition of the sum of three elements, where x_i means a term of the form $\varphi(x, i)$ for some $\varphi \in \Phi_2$, $x, i \in V$, $s \in \Phi_1$, $0, 1, 3 \in \Phi_o$. This example shows that the notion of generalized terms is natural. In the introduced language we shall give two examples of generalized terms.

Example 5.

1. $[x/x + (y \cdot z)](z - (x \cdot y))$
2. $*[x < y[x/x + 1]](x + y) + \veebar[x = y[u/x - 1][z/2]]((x + u) - z)$. $\square$

By $\tau(\tau_1/\tau_2)$ for $\tau_1, \tau_2, \tau \in T'$ we denote the expression obtained from τ by simultaneously replacing all occurrences of the generalized term τ_1 in τ by τ_2.

For further considerations we need the notion of the length of programs, generalized terms and generalized formulas.

Definition 10. *The length len of the expression will be defined as follows:*

(i) $len(\eta) = 1$ *for any* $\eta \in V \cup V_o \cup S_o \cup \Phi_o \cup \{TRUE, FALSE\}$.

(ii) *If φ is an n-argument function symbol or ρ is an n-argument predicate symbol and $\tau_1, \ldots, \tau_n$ are generalized terms then* $len(\varphi(\tau_1, \ldots, \tau_n)) = len(\rho(\tau_1, \ldots, \tau_n)) = len(\tau_1) + \ldots + len(\tau_n) + 1$.

(iii) *If the generalized formula η is of the form: $\alpha \wedge \beta$, $\alpha \vee \beta$ or $\alpha \to \beta$ then* $len(\eta) = len(\alpha) + len(\beta) + 1$, *if* $\eta = \neg \alpha$ *then* $len(\eta) = len(\alpha) + 1$, *if* $\eta = \forall_x \alpha$ *then* $len(\eta) = len(\alpha) + 1$.

(iv) *If the expression is of the form $K\eta$ where η is a generalized term or a generalized formula then* $len(K\eta) = len(K) + len(\eta)$.

(v) *If* $\eta \bigcup K\alpha$ *or* $\eta = \bigcap K\alpha$ *then* $len(\eta) = len(K) + len(\alpha) + 1$.

(vi) *If K, M are programs and α is a classical open formula then:*
$len([KM]) = len(K) + len(M) + 1$,
$len(\veebar[\alpha\, K\, M]) = len(\alpha) + len(K) + len(M) + 1$,
$len(*[\alpha\, K] = len(\alpha) + len(K) + 1$. $\square$

Moreover we define the set of all subterms of the term τ for any $\tau \in T'$.

Definition 11. *The function* $\varsigma: T' \to P(T')$ *is defined as follows:*

(i) $\varsigma(\tau) = \{\tau\}$ *for* $\tau \in V$,

(ii) $\varsigma(\varphi(\tau_1, \ldots, \tau_n)) = \varsigma(\tau_1) \cup \ldots \cup \varsigma(\tau_n) \cup \{\varphi(\tau_1, \ldots, \tau_n)\}$ *for any* $\varphi \in \Phi_n$, $n \in N$ *and* $\tau_1, \ldots, \tau_n \in T'$,

(iii) $\varsigma(K\tau) = \varsigma(\tau) \cup \{K\tau\}$ *for* $K \in S$ *and* $\tau \in T'$. $\square$

Now we introduce the operation χ defined on $T' \cup F'$ which enables us to reduce any generalized term of the form $\varphi(\tau_1, ..., \tau_n) \in T' \setminus T_o$ where $\varphi \in \Phi_n$, $\tau_1, ..., \tau_n \in T'$ and any generalized formula of the form $\rho(\tau_1, ..., \tau_n)$, where $\rho \in P_n$, $n \in N$, $\tau_1, ..., \tau_n \in T'$ to the form $K_1 ... K_m \varphi(\tau'_1, .., \tau'_n)$ or $K_1 ... K_m \rho(\tau'_1, ..., \tau'_n)$ respectively where $K_1, ..., K_m \in S$ and $\tau'_1, ..., \tau'_n \in T_o$. Therefore the operation χ changes generalized terms (generalized formulas) of the form $\varphi(\tau_1, ..., \tau_n)$ $(\rho(\tau_1, ..., \tau_n))$ by transporting all programs inside $\tau_1, ..., \tau_n$ to the left side of the expression φ or ρ.

Definition 12. *For every generalized term and every generalized formula the operation χ is defined as follows:*

(1) $\chi(\eta) = \eta$ *for* $\eta \in T_o \cup F_o$.
(2) $\chi(K\tau) = K\chi(\tau)$ *for* $K \in S$ *and* $\tau \in T'$.
(3) *If $\tau \in T' \setminus T_o$ is of the form $\varphi(\tau_1, ..., \tau_k)$ where $\varphi \in \Phi_k$, $k \in N$, $\tau_1, ..., \tau_k \in T'$, $\vartheta(\tau) = \{x_1, ..., x_n, a_1, ..., a_m\}$ and i is the smallest of the numbers $j \leq k$ such that $\tau_j \notin T_o$ and $K\tau'$ is an element of the set $\{M\tau'' \in T : M\tau'' \in \varsigma(\tau_i)$ and there is no element $M_1 t \in \varsigma(\tau_i)$ such that $len(M\tau'') \leq len(M_1 t)$ and $M\tau'' \neq M_1 t\}$ and moreover if $K\tau'$ is the earliest element of the set T, ordered linearly by a certain ordering relation, then we put*

$$\chi(\tau) = [s^{-1}\overline{sK}]\,\chi(\varphi(\tau_1, ..., \tau_{i-1}, \tau_i(K\tau'/\overline{s\tau'}), \tau_{i+1}, ..., \tau_k)),$$

where $s = [x_1/y_1, ..., x_n/y_n, a_1/b_1, ..., a_m/b_m]$ and $y_1, ..., y_n, b_1, ..., b_m$ denote the first, different individual and propositional variables not belonging to $\{x_1, ..., x_n, a_1, ..., a_m\}$ in the linearly ordered set $V \cup V_o$. The assignment instruction s^{-1} is inverse to s.
(4) *If $\alpha \in F' \setminus F_o$ is of the form $\rho(\tau_1, ..., \tau_k)$ then*

$$\chi(\alpha) = [s^{-1}\overline{sK}]\,\chi(\rho(\tau_1, ..., \tau_{i-1}, \tau_i(K\tau'/\overline{s\tau'}), \tau_{i+1}, ..., \tau_k))$$

where $\rho \in P_k$, $k \in N$, $\tau_1, ..., \tau_k \in T'$, $\vartheta(\tau) = \{x_1, ..., x_n, a_1, ..., a_m\}$ and s, i, $K\tau'$ are defined analogously as in (3).
(5) *If $\alpha \in F'$ and $K \in S$ then $\chi(K\alpha) = K\chi(\alpha)$, $\chi(\bigcup K\alpha) = \bigcup K\chi(\alpha)$ and $\chi(\bigcap K\alpha) = \bigcap K\chi(\alpha)$.*
(6) $\chi(\alpha \bullet \beta) = \chi(\alpha) \bullet \chi(\beta)$ *and* $\chi(\neg\alpha) = \neg\chi(\alpha)$ *for any* $a, \beta \in F'$ *and* $\bullet \in \{\wedge, \vee, \rightarrow\}$. $\square$

Using this function χ we add a new axiom to the set of axioms Ax
A27 $\rho(\tau_1, ..., \tau_n) \equiv \chi(\rho(\tau_1, ..., \tau_n))$

and we denote this new extended set of axioms by A_χ.

We add in Definition 6 the new condition of the form:

(c) $H(\rho(\tau_1, \ldots, \tau_n)) = H(\chi(\rho(\tau_1, \ldots, \tau_n)))$

for every $\rho(\tau_1, \ldots, \tau_n) \in F' \setminus F_o$ *where* $\rho \in P_n$, $n \in N$ *and* $\tau_1, \ldots, \tau_n \in \mathrm{T}'$.

Moreover we have to change in Definition 7the notion of program-substitution $p : F' \to F'$ *as follows:*

(d)
$$p(\alpha) = \begin{cases} h^e(\alpha) & \text{for } \alpha \in F_o \\ s^{\chi(\alpha)} H(\alpha) & \text{for } \alpha \in F' \subset F_o. \end{cases}$$

This new set of program-substitutions will be denoted by Sb_χ.

Now we present an example to explain the effects of program-substitution from Sb_χ.

Example 6. Let z, u denote the first, different individual variables of the set $V \setminus \{x, y\}$ where $a \in V_o$ and $x \neq y$. It can be easily seen that for $e \in \mathcal{E}_g$ and for $p \in Sb_\chi$ such that p is defined as in **(d)** we get

$$\begin{aligned} p(a \wedge \rho([x/y]x)) &= [g(a)/e(a)]\, H(a \wedge \rho([x/y]x)) \\ &= [g(a)/e(a)]\,(g(a) \wedge H(\chi(\rho([x/y]x)))) \\ &= [g(a)/e(a)]\,(g(a) \wedge H([[z/x, u/y]\ \overline{[x/z, y/u][x/y]]}\ \rho\ \overline{([x/z, y/u]x)})) \\ &= [g(a)/e(a)]\,(g(a) \wedge [[g(z)/g(x), g(u)/g(y)]\ \ [g(z)/g(u)]]\, e(\rho(z)). \end{aligned}$$

By Lemma 1 (i) and by applying **A14, A23** we conclude that the generalized formula

$$[g(a)/e(a)]\,(g(a) \wedge [[g(z)/g(x), g(u)/g(y)]\,[g(z)/g(u)]]\, e(\rho(z)))$$

is equivalent to the formula of the form

$$e(a) \wedge \overline{[g(z)/g(x), g(u)/g(y)]\, \overline{([g(z)/g(u)]\, e(\rho(z)))}}.$$

Since $e \in \mathcal{E}_\wp$ then two classical open formulas

$$e(a) \wedge \overline{[g(z)/g(x), g(u)/g(y)]\, \overline{([g(z)/g(u)]\, e(\rho(z)))}} \text{ and } e(a) \wedge e(\rho(y))$$

are equivalent.

So we can say intuitively that p transforms a; $\rho(y)$ into $e(a)$; $e(\rho(y))$ respectively. ■

One can observe that if we change the symbols s^α, s^β, Sb *in the proof of Theorem 2 for* $s^{\chi(\alpha)}$, $s^{\chi(\beta)}$, Sb_χ *respectively and if we extend the meaning of the symbol* H *by* **(c)** *and the meaning of the notion of program-substitution* p *by* **(d)** *then we get*

Theorem 4. *Algorithmic logic with generalized terms is closed under program-substitution from* Sb_χ. □

In the sequel we extend Definition 9 assuming that

(16) $< s\chi(\rho(\tau_1, ..., \tau_n)), s\rho(\tau_1, ..., \tau_n)> \in X$ *for* s *being a sequence of assignment instructions* $s_1 ... s_k$, $k \in N$, $\rho \in P_n$, $n \in N$, $\tau_1, ..., \tau_n \in T'$ *and for* $\rho(\tau_1, ..., \tau_n) \in F' \setminus F_o$.

Therefore we obtain the relation $<$ *defined on the set of generalized formulas of the language with generalized terms. These new definitions enable us to formulate some version of Lemma 3.*

Lemma 4. *For any generalized formulas* $\alpha, \beta \in F'$ *and any* $g \in G$, $e \in \mathscr{E}_g$, (e, g)*-function* u *and an endomorphism* $H : F' \to F'$ *such that* $H/F_o = h^u$, *if* $\vartheta(\chi(\alpha)) \cap V_o \subset \vartheta(\beta)$, *then* $\vdash s^\beta H(\alpha) \equiv s^{\chi(\alpha)} H(\alpha) \equiv s^{\chi(\alpha)} H(\alpha)$, *where* s^β *and* $s^{\chi(\alpha)}$ *are designated by* $<g, e>$. □

By applying Lemma 4 we can prove the theorem analogous to the Theorem 3:

For every program-substitution $p \in Sb_\chi$ *and for all generalized formulas* α, $\beta \in F'$:

(i) $\vdash p(\alpha \to \beta) \equiv (p(\alpha) \to p(\beta))$,
(ii) $\vdash p(\alpha \wedge \beta) \equiv (p(\alpha) \wedge p(\beta))$,
(iii) $\vdash p(\alpha \vee \beta) \equiv (p(\alpha) \vee p(\beta))$,
(iv) $\vdash p(\neg\alpha) \equiv \neg p(\alpha)$. □

3.5 Program-substitution in the language $\mathscr{L}''$ and $\mathscr{L}_\square$

The above considerations can be generalized for the language of the extended algorithmic logic of the first order with classical quantifiers $\mathscr{L}''$ *(L Banachowski [1]) and for the language of algorithmic logic with non-deterministic programs (G. Mirkowska [60], [61]).*

To get the set of axioms of $\mathscr{L}''$ *we add some new forms of axioms to the set of axioms* Ax:

Q27 $s(\exists_x \alpha) \equiv \exists_y s([x/y]\alpha)$ *for* $y \notin \vartheta(s\alpha)$,
Q28 $[x/\tau]\alpha \to \exists_x \alpha$ *where* $\alpha \in F_V$, $x \in V$ *and* $\tau \in T_o$,
Q29 $\forall_x \alpha \equiv \neg \exists_x(\neg\alpha)$.

We admit r_o, r_3 *as the rules of inference and two rules of the scheme:*

$$r_4 : \frac{\alpha \to \beta}{K\alpha \to K\beta}, \quad r_5 : \frac{[x/y]\,\alpha \to \beta}{\exists_x \alpha \to \beta},$$

where $\alpha, \beta \in F_V$, $K \in S$, $x, y \in V$ *and* $y \notin \vartheta(\alpha \wedge \beta)$.

Let C_E be the consequence operation of the extended algorithmic logic with quantifiers *based on the set of rules $\{r_o, r_3, r_4, r_5\}$ and defined by Definition 1. Any formula of the set $C_E(\emptyset)$ will be called a thesis of algorithmic logic with quantifiers and $\alpha \in C_E(\emptyset)$ will be denoted by $\models \alpha$.* $\square$

Now we shall introduce the language of algorithmic logic with non-deterministic programs. *There are many reasons, that motivate and justify studies of algorithmic properties of non-deterministic programs, cf. D. Harel and V. Pratt [39], G. Mirkowska [60], [61], S. Radziszowski [77]. We have auxiliary symbols: $\cup$, $\square$, $\diamond$. Non-deterministic programs are constructed with the new program connective $\cup$ (non-deterministic choice) and are denoted by $[K_1 \cup K_2]$. The programs without the symbol $\cup$ will be called deterministic. In this language the set of programs will be denoted by $S_\square$. We have new generalized formulas in the language of non-deterministic programs $\mathcal{L}_\square$. To the formation rules describing the set $F_\forall$ we add new formation rules and we change the symbol of the set of generalized formulas and denote it by $F_\square$:*

(1) *If $\alpha \in F_\square$ and $K \in S_\square$ is a deterministic program then $K\alpha \in F_\square$,*
(2) *If $\alpha \in F_\square$ and K is a non-deterministic program then $\square K\alpha$, $\diamond K\alpha$, $\square \bigcup K\alpha$, $\square \bigcap K\alpha$, $\diamond \bigcup K\alpha$, $\diamond \bigcap K\alpha \in F_\square$.*

By a configuration *in the realization $\mathfrak{R}$ we shall mean any ordered pair $<v, K_1; ...; K_n>$ where $v \in W$ is a valuation and $K_1, ..., K_n \in S_\square$.* $\square$

Let $\rightarrow_\mathfrak{R}$ be the least binary relation in the set of all configurations such that the following conditions hold:

(1) *If s is of the form* (a) *then $<v, s; rest> \rightarrow_\mathfrak{R} <v', rest>$ where v' is a valuation such that $v'(x_i) = \tau_{i\mathfrak{R}}$ for $1 \leq i \leq n$ and $v'(z) = v(z)$ for $z \in (V \cup V_o) \setminus \{x_1, ..., x_n\}$,*
(2) $<v, [K \cup M]; rest> \rightarrow_\mathfrak{R} <v, K; rest>$,
(3) $<v, [K \cup M]; rest> \rightarrow_\mathfrak{R} <v, M; rest>$,
(4) $<v, \veebar[\alpha KM]; rest> \rightarrow_\mathfrak{R} <v, K; rest>$ *iff* $\alpha_\mathfrak{R}(v) = \bigvee_o$,
(5) $<v, \veebar[\alpha KM]; rest> \rightarrow_\mathfrak{R} <v, M; rest>$ *iff* $\alpha_\mathfrak{R}(v) = \bigwedge_o$,
(6) $<v, [KM]; rest> \rightarrow_\mathfrak{R} <v, K, M; rest>$,
(7) $<v, *[\alpha K]; rest> \rightarrow_\mathfrak{R} <v, rest>$ *iff* $\alpha_\mathfrak{R}(v) = \bigwedge_o$,
(8) $<v, *[\alpha K]; rest> \rightarrow_\mathfrak{R} <v, K; *[\alpha K]; rest>$ *iff* $\alpha_\mathfrak{R}(v) = \bigvee_o$. $\square$

Let $I \subset \mathcal{N}$ and for any $n, m \in \mathcal{N}$, the following condition holds: if $n \leq m$ and $m \in I$, then $n \in I$.

The sequence $(c_i)_{i \in I}$ of configurations is called a computation of the program $K \in S_\square$ in the realization $\mathfrak{R}$ and at the valuation v iff for any $i, j \in I$: if $j = i + 1$ then $c_i \rightarrow_\mathfrak{R} c_j$ and $c_o = <v, K>$. $\square$

If the sequence $(c_i)_{i\in I}$ is finite, i.e. if it is a sequence $c_1, ..., c_n$ and $c_n = \langle v', \emptyset \rangle$, then the valuation v' is called the result of the computation of the program K in the realization $\mathcal{R}$ at the valuation $v \in W$. □

The set of all results of the computation of the program $K \in S_\Box$ in the realization $\mathcal{R}$ at the valuation $v \in W$ will be denoted by $K_\mathcal{R}(v)$. So if $K \in S_\Box$ is a deterministic program, then $K_\mathcal{R}(v)$ is at most a one-element set.

Let $K' \in S_\Box$ be a deterministic program and $K \in S_\Box$. We put

$$(K'\alpha)_\mathcal{R}(v) = \begin{cases} \alpha_\mathcal{R}(v') & \textit{iff } v' \in K'_\mathcal{R}(v) \\ \bigwedge_o & \textit{in the opposite case} \end{cases}$$

$(\Box K\alpha)_\mathcal{R}(v) = \bigvee_o$ iff all computations of K in the realization $\mathcal{R}$ and at the valuation $v \in W$ are finite and for all $v \in W$, if $v' \in K_\mathcal{R}(v)$ then $\alpha_\mathcal{R}(v') = \bigvee_o$,
$(\Diamond K\alpha)_\mathcal{R}(v) = \bigvee_o$ iff K has a finite computation in the realization $\mathcal{R}$ and at the valuation $v \in W$ and there exist $v' \in K_\mathcal{R}(v)$ such that $\alpha_\mathcal{R}(v') = \bigvee_o$,
$(\Box \bigcup K\alpha)_\mathcal{R}(v) = \sup\{((\Box K)^i\alpha)_\mathcal{R}(v) : i \in \mathcal{N}\}$,
$(\Diamond \bigcup K\alpha)_\mathcal{R}(v) = \sup\{((\Diamond K)^i\alpha)_\mathcal{R}(v) : i \in \mathcal{N}\}$,
$(\bigcup K'\alpha)_\mathcal{R}(v) = \sup\{(K'^i\alpha)_\mathcal{R}(v) : i \in \mathcal{N}\}$,
$(\bigcap K'\alpha)_\mathcal{R}(v) = \inf\{(K'^i\alpha)_\mathcal{R}(v) : i \in \mathcal{N}\}$,
$(\Box \bigcap K\alpha)_\mathcal{R}(v) = \inf\{((\Box K)^i\alpha)_\mathcal{R}(v) : i \in \mathcal{N}\}$,
$(\Diamond \bigcap K\alpha)_\mathcal{R}(v) = \inf\{((\Diamond K)^i\alpha)_\mathcal{R}(v) : i \in \mathcal{N}\}$,

where $(\Box K)^i\alpha$ (and analogously $(\Diamond K)^i\alpha$) denotes the formula:

$$\underbrace{\Box K(\Box K(...(\Box K\alpha)...))}_{i-times}, \quad \underbrace{(\Diamond K(\Diamond K(...(\Diamond K\alpha)...)))}_{i-times}. \ \Box$$

Hence for example we get

$$*[\delta K]_\mathcal{R}(v) = \{v' \in W : \exists_{v_o,...,v_n} \forall_{0<i\leq n}(v_i \in K_\mathcal{R}^i(v) \textit{ and } v_o = v \textit{ and}$$
$$\delta_\mathcal{R}(v_n) = \bigwedge_o \textit{ and } v_n = v')\},$$

$$[K\, M]_\mathcal{R}(v) = \{v' \in W : \exists_{v''}(v'' \in K_\mathcal{R}(v) \textit{ and } v' \in M_\mathcal{R}(v''))\},$$

$$\veebar[\delta\ KM]_\mathcal{R}(v) = \begin{cases} K_\mathcal{R}(v) & \textit{if } \delta_\mathcal{R}(v) = \bigvee_o \\ M_\mathcal{R}(v) & \textit{if } \delta_\mathcal{R}(v) = \bigwedge_o, \end{cases}$$

$$[K \cup M]_\mathcal{R}(v) = K_\mathcal{R}(v) \cup M_\mathcal{R}(v).$$

Now we present the axioms and rules of the algorithmic logic with non-deterministic programs from $\mathscr{L}_{\square}$ (G. Mirkowska [61]): **A1-A14, A26, Q27-Q29** *and* **A15-A24** *for deterministic programs and moreover for* $\otimes \in \{\square, \diamondsuit\}$:

N1 $\square K(\alpha \wedge \beta) \equiv (\square K\alpha \wedge \square K\beta)$
N2 $\diamondsuit K(\alpha \vee \beta) \equiv (\diamondsuit K\alpha \vee \diamondsuit K\beta)$
N3 $\otimes[K\, M]\alpha \equiv \otimes K(\otimes M\alpha)$
N4 $\otimes \veebar[\delta\, K\, M]\alpha \equiv ((\delta \wedge \otimes K\alpha) \vee (\neg\delta \wedge \otimes M\alpha))$
N5 $*[\delta\, K']\alpha \equiv ((\neg\delta \wedge \alpha) \vee (\delta \wedge K'(*[\delta\, K']\alpha)))$
N6 $\otimes *[\delta\, K]\alpha \equiv ((\neg\delta \wedge \alpha) \vee (\otimes K(\otimes *[\delta\, K]\alpha)))$
N7 $\diamondsuit[K \cup M]\alpha \equiv (\diamondsuit K\alpha \vee \diamondsuit M\alpha)$
N8 $\square[K \cup M]\alpha \equiv (\square K\alpha \wedge \square M\alpha)$
N9 $\otimes \bigcup K\alpha \equiv (\alpha \vee \otimes \bigcup K(\otimes K\alpha))$
N10 $\otimes \bigcap K\alpha \equiv (\alpha \vee \otimes \bigcap K(\otimes K\alpha))$
N11 $\otimes K'\alpha \equiv K'\alpha$.

We adopt the following rules:
r_o, r_5 *and* r_2, r_3, r_4 *for deterministic programs and moreover*

$$r_6 : \frac{\alpha \to \beta}{\otimes K\alpha \to \otimes K\beta}, \qquad r_7 : \frac{\{(s \veebar[\delta\, K'[\,]]^i(\neg\delta \wedge \alpha)) \to \beta : i \in \mathcal{N}\}}{(s*[\delta\, K']\alpha) \to \beta}$$

$$r_8 : \frac{\{(s\otimes \veebar[\delta\, K[\,]]^i(\neg\delta \wedge \alpha)) \to \beta : i \in \mathcal{N}\}}{(s\otimes *[\delta\, K]\alpha) \to \beta}, \qquad r_9 : \frac{\{(s\otimes K^i\alpha) \to \beta : i \in \mathcal{N}\}}{(s\otimes \bigcup K\alpha) \to \beta},$$

$$r_{10} : \frac{\{\beta \to s\otimes K^i\alpha : i \in \mathcal{N}\}}{\beta \to s\otimes \bigcap K\alpha},$$

where $\otimes \in \{\square, \diamondsuit\}$.

These rules and axioms define the consequence operation of the algorithmic logic with non-deterministic programs denoted by $C_{\square}$ *(G. Mirkowska [61]).*

The set of program-substitutions $Sb_{\forall}$ *is defined in the extended algorithmic logic with quantifiers analogously as in AL though defining the function* $\bar{H}$ *(see Definition 6) we put*

$$H(\exists_x \alpha) = \exists_{\varrho(x)} H(\alpha) \text{ and } H(\forall_x \alpha) = \forall_{\varrho(x)} H(\alpha)$$

where $x \in V$ *and* $\alpha \in F_{\forall}$.

Moreover in the algorithmic logic with non-deterministic programs the set $Sb_{\Box}$ *is defined analogously as in* $\mathscr{L}''$ *but we put additionally*

$$H(DK\alpha) = DK_g^e H(\alpha)$$

for any $D \in \{\Diamond, \Diamond\bigcup, \Diamond\bigcap, \Box, \Box\bigcup, \Box\bigcap\}$ *and* $\alpha \in F_{\Box}$.

Lemma 5.

(i) $g(V \setminus \vartheta(\alpha)) \cap \vartheta(h^u(\alpha)) = \emptyset$ *for every* $\alpha \in F_o$,
(ii) $g(V \setminus \vartheta(K)) \cap \vartheta(K_g^e) = \emptyset$ *for every* $K \in S_{\Box}$,
(iii) *For every generalized formula* α *and for every individual variable* $y \in V$, *if* $y \notin \vartheta(\alpha)$, *then* $g(y) \notin \vartheta H(\alpha)$), *where* $g \in G$ *and* $e \in \mathscr{E}_g$ $\Box$.

Theorem 5. *The extended algorithmic logic with quantifiers and the algorithmic logic with non-deterministic programs are closed under program-substitutions i.e.* $p(C_E(\emptyset)) \subset C_E(\emptyset)$ *and* $p(C_{\Box}(\emptyset)) \subset C_{\Box}(\emptyset)$ *for every program-substitution* $p \in Sb_{\forall}$ *or* $p \in Sb_{\Box}$ respectively. $\Box$

Let us consider the set X' *of pairs* $<\alpha_1, \alpha_2>$ *in algorithmic logic with quantifiers such that* α_1 *is equal to* α_2 *or* $<\alpha_1, \alpha_2>$ *is one of the form from Definition 9 or additionally of the form*
(16) $<s_1 \dots s_n[x/\tau]\alpha, s_1 \dots s_n \exists_x \alpha>$ *for* α_1, α_2, $\alpha \in F_{\forall}$, $s_1, \dots, s_n \in S_o$, $n \in N$, $x \in V$ *and* $\tau \in T_o$.
We define the binary relation $<'$ *on* $F_{\forall}$ *in the extended algorithmic logic with quantifiers for any* α, $\beta \in F_{\forall}$:
$\alpha <' \beta$ *iff there exist* $\alpha_1, \dots, \alpha_n \in F_{\forall}$ *such that* $\alpha_1 = \alpha$, $\alpha_n = \beta$ *and for every* $i \in \{1, \dots n-1\}$ *the pair* $<\alpha_i, \alpha_{i+1}>$ *is an element of* X'.

Let us notice that the relation $<'$ *is an ordering in* $F_{\forall}$ *such that any nonempty subset* $Z \subset F_{\forall}$ *contains a minimal element. The above binary relation can be defined on* $F_{\Box}$. *For further considerations we shall need.*

Lemma 6. *For any generalized formulas* $\alpha, \beta \in F_{\forall}(F_{\Box})$: *if* $\vartheta(\alpha) \cap V_0 \subset \vartheta(\beta)$ *then* $s^{\beta}H(\alpha) \leftrightarrow s^{\alpha}H(\alpha)$ *is a thesis of* $C_E(\emptyset)$ $(C_{\Box}(\emptyset))$. $\Box$

Proof. For simplicity we shall prove this lemma only for the language of the extended algorithmic logic with quantifiers. The proof is by induction on the relation $<'$ *defined on* $F_{\forall}$. *Since the proof is analogous to the proof of Lemma 3, we consider only the case* $\alpha = \exists_x \lambda$ *for some* $x \in V$ *and for some generalized formula* λ.

Since $[x/\tau]\lambda <' \exists_x \lambda$ *and* $\vartheta(\lambda) \cap V_0 = \vartheta([x/\tau]\lambda) \cap V_0 = \vartheta(\alpha) \cap V_0$ *for every* $\tau \in T_o$, *we get*

(1) $\vdash s^{\beta}H([x/\tau]\lambda) \leftrightarrow s^{\alpha}H([x/\tau]\lambda)$ *by the equality* $s^{[x/\tau]\lambda} = s^{\alpha}$ *and by the inductive hypothesis for every* $\tau \in T_o$.

The inclusion $H(C_E(\emptyset)) \subset C_E(\emptyset)$ *holds and the rule of the scheme* $\frac{\beta}{s\beta}$ *is a derivable rule of the consequence operation* C_E, *so by the axiom* $[x/\tau]\lambda \to \exists_x\lambda$ *we conclude that*

(2) $\vdash s^{\alpha}H([x/\tau]\lambda) \to s^{\alpha}H(\alpha)$ *for every* $\tau \in T_o$.

According to (1) and (2) we obtain

(3) $\vdash s^{\beta}[g(x)/g'(\tau)]H(\lambda) \to s^{\alpha}H(\alpha)$ *for every* $\tau \in T_o$.

We shall use the abbreviation $\mathcal{Q} = g(V) \setminus \vartheta(s^{\beta}s^{\alpha}\,TRUE)$. $\mathcal{Q}$ *is an infinite and enumerable set whereas* $g(\vartheta(\lambda) \cup \{x\})$ *is a finite set, so there exists* $z \in \mathcal{Q} \setminus g(\vartheta(\lambda) \cup \{x\})$. *Hence and by the definition of* $\mathcal{Q}$ *there exists* $y \in V$ *such that* $z = g(y)$ *and* $g(y) \notin \vartheta(s^{\beta}) \cup \vartheta(s^{\alpha})$. *Since* $g(y) \notin g(\vartheta(\lambda) \cup \{x\})$, *we get* $y \notin \vartheta(\lambda)$ *and* $g(y) \neq g(x)$.

According to the Lemma 5 (iii) we get $g(y) \notin \vartheta(H(\lambda))$. *Hence* $g(y) \notin \vartheta(H(\lambda)) \cup \{g(x)\} \cup \vartheta(s^{\beta}) \cup \vartheta(s^{\alpha})$.

Putting in (3) $\tau = y$ *and using the rule of the scheme* $\frac{([x/y]\beta) \to \delta}{(\exists_x\beta) \to \delta}$ *we conclude that*

(4) $\vdash s^{\beta}H(\alpha) \to s^{\alpha}H(\alpha)$ *for* $y \notin \vartheta(\alpha \cdot \beta)$.

By the same argumentation as used in (2) we get $\vdash s^{\beta}[g(x)/g'(\tau)]H(\lambda) \to s^{\beta}H(\alpha)$. *Hence and by (1) we obtain*

(5) $\vdash s^{\alpha}[g(x)/g'(\tau)]H(\lambda) \to s^{\beta}H(\alpha)$.

Simultaneously by similar argumentation as before we find a special element $y \in V$ *and then putting in (5)* $\tau = y$ *and using the above-mentioned rule we get*

(6) $\vdash s^{\alpha}H(\alpha) \to s^{\beta}H(\alpha)$.

On the other hand by (4) and (6) we conclude that

(7) $\vdash s^{\alpha}H(\alpha) \leftrightarrow s^{\alpha}H(\alpha)$. ■

Chapter 4

Algorithmic structural completeness

4.1 The problem of completeness of C_{R_*}

In this chapter we introduce the notion of the algorithmic structural completeness and we shall prove property for the consequence operation of AL. At first we shall consider the substitution rule and the structural rules. Next we shall study interrelation between all structural, finitary and admissible rules on one hand and derivable rules on the other hand.

By a *substitution rule* r_* we mean the rule of the form: $\frac{\alpha}{p(\alpha)}$ where p is a program-substitution. Assume the following abbreviation: $R_* = R \cup \{r_*\}$. □

It will appear that the substitution rule allows us to examine deeply algorithmic properties of formulas and programs of AL.

We shall say that *a rule r is structural* iff $<p(H), p(\alpha)> \in r$ for every sequent $<X, \alpha> \in r$ and for every program-substitution $p \in Sb$. □

We recall two definitions:

A generalized formula α is an element of the set $\mathcal{J}$ iff there exists a classical formula β (i.e. a formula without programs) such that α and β are equivalent.

A rule r is finitary iff for every sequent $<X, \alpha> \in r$ the set X is finite and $X \cup \{\alpha\} \subset \mathcal{J}$.

The consequence operation C is *algorithmically structurally complete* iff every structural, finitary and admissible rule is derivable in C. □

Theorem 2 allows us to strengthen the consequence operation C_R by substitution rule and to examine the C_{R_*} — consequence operation. Obviously $C_{R_*}(\emptyset) = C_R(\emptyset)$. At first we shall solve the problem of completeness of C_{R_*} and next we shall prove that the consequence operation of algorithmic logic strengthened by the substitution rule is algorithmically structurally complete.

Definition 13. *Let $x_o \in V$. For any one-one mapping h_o of the set V into $V \setminus \{x_o\}$ we define a function h on $T_o \cup S \cup F$ as follows:*

(1) $h(x) = h_o(x)$ *for every* $x \in V$,
(2) $h(\varphi(\tau_1, ..., \tau_n)) = \varphi(h(\tau_1), ..., h(\tau_n))$ *for any* $\varphi \in \Phi_n$, $n \in N$ *and* $\tau_1, ..., \tau_n \in T_o$,
(3) $h(\varphi) = \varphi$ *for any* $\varphi \in \Phi_o$,
(4) $h(\alpha) = \alpha$ *for every* $\alpha \in V_o \cup \{TRUE, FALSE\}$,
(5) $h(\rho(\tau_1, ..., \tau_n)) = \rho(x_o, ..., x_o)$ *for any formula* $\rho(\tau_1, ..., \tau_n) \in E$,
(6) $h(\alpha \bullet \beta) = h(\alpha) \bullet h(\beta)$ *and* $h(\neg\alpha) = \neg h(\alpha)$ *for any generalized formulas* α, β *and* $\bullet \in \{\wedge, \vee, \rightarrow\}$,
(7) $h(s) = \begin{cases} [\,] & \text{if } s \text{ is of the form } [x_1/\tau_1, ..., x_n/\tau_n] \\ [a_1/h(\alpha_1), ..., a_m/h(\alpha_m)] & \text{if } s \text{ is of the form (a)}, \end{cases}$
(8) $h([K\,M]) = [h(K)\,h(M)]$,
(9) $h(\veebar[\delta KM]) = \veebar[h(\delta)\,h(K)\,h(M)]$,
(10) $h(*[\delta\,K]) = *[h(\delta)\,h(K)]$,
(11) $h(K\alpha) = h(K)h(\alpha)$,
(12) $h(\bigcup K\alpha) = \bigcup h(K)h(\alpha)$ and $h(\bigcap K\alpha) = \bigcap h(K)h(\alpha)$ □.

Theorem 6. *The consequence operation C_{R_*} is incomplete.* □

Proof. At first we shall prove the following inclusion:

(e) $h(C_{R_*}(\{\rho(x) \rightarrow \rho(y)\})) \subset C_R(\emptyset)$.

By Definition 1. we know that $C_R(X) = \bigcup\{C_R^\gamma(X) : \gamma < \Omega\}$. We shall prove that $h(C_{R_*}^\gamma(\{\rho(x) \rightarrow \rho(y)\})) \subset C_R(\emptyset)$ for every γ.

Let γ be the least ordinal number i.e. $\gamma = 0$. Since $\overline{h(s)h(\delta)} = h(\overline{s\delta})$ for every $s \in S_o$ and for every $\delta \in F_o$, the value resulted from applying h to the axiom A14 is the thesis. Hence

$$h(C_{R_*}^0(\{\rho(x) \rightarrow \rho(y)\})) \subset C_R(\emptyset).$$

We assume inductively that $h(C_{R_*}^\mu(\{\rho(x) \rightarrow \rho(y)\})) \subset C_R(\emptyset)$ for every ordinal number $\mu < \gamma$.

If γ is a limit ordinal number, then by the inductive hypothesis we get

$$h(C_{R_*}^\gamma(\{\rho(x) \rightarrow \rho(y)\})) \subset C_R(\emptyset).$$

Now suppose that $\gamma = \mu_0 + 1$ for some ordinal number μ_0. Let $\alpha \in h(C^{\gamma}_{R_*}(\{\rho(x) \to \rho(y)\}))$. Hence and by Definition 1 we get

$$\alpha \in h(C^{\mu_0}_{R_*}(\{\rho(x) \to \rho(y)\})).$$

Then according to the inductive hypothesis $\vdash \alpha$ or there exist $X \subset (C^{\mu_0}_{R_*}(\{\rho(x) \to \rho(y)\})$, $\beta \in F$ and $r \in R_*$ such that $\alpha = h(\beta)$ and $<X, \beta> \in r$.

If $r \neq r_*$ then $<h(X), h(\beta)> \in r$ and by the inductive hypothesis $h(X) \subset C_R(\emptyset)$. Using the rule r we get $\vdash \alpha$. Hence

(1) $h(C_R(\emptyset)) \subset C_R(\emptyset)$.

If $r = r_*$, then $X = \{\lambda\}$ for some $\lambda \in F$ and by the inductive hypothesis $\vdash h(\lambda)$. Since $<\{\lambda\}, \beta> \in r_*$, there exists $p \in Sb$ such that $\beta = p(\lambda)$. Thus $\alpha = h(p(\lambda))$. As we know for any $\eta \in F$

$$p(\eta) = \begin{cases} h^e(\eta) & \text{for } \eta \in F_o \\ s^\eta H(\eta) & \text{for } \eta \in F \setminus F_o \end{cases}$$

where $g \in G$, $e \in \mathscr{E}_g$ and H fulfills the conditions from Definition 6, so $H/F_o = h^u$ for (e, g)*-function* u.

For further considerations some functions will be defined and their properties will be thoroughly analysed.

Let h^{e_*} be an endomorphism on F_o such that for any $\delta \in At$

$$e_*(\delta) = \begin{cases} h(p(\delta_1)) & \text{if } \delta = h(\delta_1) \quad \text{for some } \delta_1 \in At \\ h(p(\delta)) & \text{otherwise} \end{cases}$$

The above definition is correct, since it is enough to show that for any classical open formula λ_1, $\lambda_2 \in F_o$:

$$\text{if} \quad h(\lambda_1) = h(\lambda_2), \quad \text{then} \quad h(p(\lambda_1)) = h(p(\lambda_2)).$$

For $\lambda_1, \lambda_2 \in V_0 \cup \{TRUE, FALSE\}$ by assumption and Definition 13 we get $\lambda_1 = \lambda_2$, which gives the thesis. If λ_1, $\lambda_2 \in E$ then

$$\lambda_1 = \overline{[x_1/\tau_1, ..., x_n/\tau_n]\ \rho(x_1, ..., x_n)}$$

and

$$\lambda_2 = \overline{[x_1/\tau'_1, ..., x_n/\tau'_n]\ \rho(x_1, ..., x_n)}$$

for some classical terms and for some n-argument predicate letter. Therefore by the definition of the set $\mathscr{E}_g$ and by Definition 13 we get

$$h(p(\lambda_1)) = h(\overline{e([x_1/\tau_1, ..., x_n/\tau_n]\ \rho(x_1, ..., x_n))})$$

$$= h(\overline{g'([x_1/\tau_1, ..., x_n/\tau_n]\ e(\rho(x_1, ..., x_n)))}).$$

Obviously for any s_1, $s \in S_o$ and any $\eta \in F_o$ if $(\vartheta(s_1) \cup \vartheta(s_2)) \cap V_0 = \emptyset$ then $h(\overline{s_1\eta}) = h(\overline{s_2\eta})$. Since $g'(s) \in S_o$ for every $s \in S_o$ and $e(\rho(x_1, ..., x_n)) \in F_o$, we get

$$h(p(\lambda_1)) = h(\overline{g'([x_1/\tau'_1, ..., x_n/\tau'_n])\ e(\rho(x_1, ..., x_n))})$$

$$= h(\overline{e([x_1/\tau'_1, ..., x_n/\tau'_n]\ \rho(x_1, ..., x_n))}) = h(p(\lambda_2)).$$

Let h^{e_1} be an endomorphism on F_o such that for every $\delta \in At$

$$e_1(\delta) = \begin{cases} h(g(\delta)) & \text{if } \delta \in V_0 \\ e_*(\delta) & \text{otherwise.} \end{cases}$$

For any program $K \in S$ we define a program K' as follows:

(i) If K is of the form (a) then we put

$$K' = [h(g(x_1))/h(g'(\tau_1)), ..., h(g(x_n))/h(g'(\tau_n)),\ g(a_1)/h^{e_1}(\alpha_1), ..., g(a_m)/h^{e_1}(\alpha_m)].$$

Obviously if $K = [\,]$, then $K' = [\,]$,

(ii) If K is one of the form $\circ[MN]$, $\veebar[\delta MN]$ or $*[\delta M]$, then K' is of the form $\circ[M'\,N']$, $\veebar[h^{e_1}(\delta)M'\,N']$ or $*[h^{e_1}(\delta)M']$ respectively.

Now we define a mapping H_1 on F in the same way as it was done for the function $H: F \to F$ from Definition 6, i.e. instead of the (e, g)-*function* u and K_g^e we put there e_1 and K' respectively.

Now assume that s^δ is determined by a couple $<h \circ g, e_*>$ for every $\delta \in F$. Let q be a mapping defined on F as follows:

$$q(\delta) = \begin{cases} h^{e_*}(\delta) & \text{if } \delta \in F_o \\ s^\delta H_1(\delta) & \text{if } \delta \in F \setminus F_o. \end{cases}$$

Since $\overline{h^{e_*}(s\rho(\tau_1, ..., \tau_n))} = \overline{h(g'(s))e_*(\rho(\tau_1, ..., \tau_n))}$ for every elementary formula $\rho(\tau_1, ..., \tau_n) \in E$ and for every $s \in S_o$, we get $h(g(V_0)) \cap \vartheta(e_1(E)) = \emptyset$. Of course $\vartheta(e_1(\delta)) \cap V \subset \{x_0\}$ for any elementary formula δ. Hence $\bar{s}' z = \overline{h(g'(s))}z$ for every $s \in S$ and every $z \in \vartheta(e_1(E))$. Thus

(2) $\overline{s' h^{e_1}(\delta)} = h^{e_1}(\bar{s}\delta)$ for every $\delta \in F_o$ and $s \in S_o$.

Moreover we get

(3) $s^{\delta} H_1(\alpha') = h^{e_*}(\alpha')$ for every δ, $\alpha' \in F_o$ such that $V_0 \cap \vartheta(\alpha') \subset \vartheta(\delta)$.

By (2) we obtain the inclusion $H_1(Ax) \subset C_R(\emptyset)$. Simultaneously $< H_1(X), H_1(\alpha') > \in r$ for every $r \in R$ and every $< X, \alpha' > \in r$. According to these considerations we get

(4) $H_1(C_R(\emptyset)) \subset C_R(\emptyset)$.

By (3), (4) and by the inductive hypothesis we get

(5) $q(C_R(\emptyset)) \subset C_R(\emptyset)$.

Using (3) and (4) we can prove by an analogous argument as used in Lemma 3 the following property:

(6) For any generalized formulas, Φ, $\Psi \in F$ if $\vartheta(\Phi) \cap V_0 \subset \vartheta(\Psi)$, then $\vdash s^{\Psi} H_1(\Phi) \leftrightarrow s^{\Phi} H_1(\Phi)$.

Similarly as in Theorem 3 by virtue of (3) and (6) we can prove that for every Φ, $\Psi \in F$:

(7) $\vdash q(\Phi \bullet \Psi) \leftrightarrow (q(\Phi) \bullet q(\Psi))$ for any $\bullet \in \{\wedge, \vee, \rightarrow\}$ and $\vdash q(\neg\Phi) \leftrightarrow \neg q(\Phi)$.

Now we are going to prove the following equivalence:

(8) $\vdash q(h(\Phi)) \leftrightarrow (h(p(\Phi))$ for every $\Phi \in F$.

If Φ is a minimal element of the relation $<$ introduced in Definition 9 then the thesis holds in this case.

Suppose that (8) holds for every generalized formula $\Phi' \in F$ such that $\Phi' < \Phi$.

Let Φ be of the form $s_1, ..., s_n\delta$ for some $\delta \in F_o$ and for some $s_1, ..., s_n \in S_o$. By A14, r_1 and (1) we get

$$\vdash h(s_1 ... s_n\delta) \leftrightarrow h(s_1 ... s_{n-1} \overline{s_n\delta}).$$

Hence, by (5) and (7) we conclude that

$$\vdash q(h(s_1 \dots s_n\delta) \leftrightarrow q(h(s_1 \dots s_{n-1} \overline{s_n\delta})).$$

Using the inductive hypothesis it follows that

$$\vdash q(h(s_1 \dots s_{n-1} \overline{s_n\delta})) \leftrightarrow h(p(s_1 \dots s_{n-1} \overline{s_n\delta})).$$

Moreover A14, r_1 and (1) allow us to get

$$\vdash h(p(s_1 \dots s_{n-1} \overline{s_n\delta})) \leftrightarrow h(p(s_1 \dots s_n\delta)).$$

Therefore the thesis holds in this case.

If Φ is either of the form $s_1 \dots s_n \circ [KM]\ \Psi$, $s_1 \dots s_n \veebar [\delta KM]\ \Psi$ or of the form $s_1 \dots s_n * [\delta K]\ \Psi$, then by A23, A24 and A25 respectively and quite similiar argumentation as used before we get the thesis.

Moreover for Φ being of the form $s_1 \dots s_m(\Psi \bullet \Psi')$ or $s_1 \dots s_m(\neg\Psi)$ for some $\bullet \in \{\wedge, \vee, \rightarrow\}$ the proof is analogous as before by using the axioms A16, A15, A19, A20, A17, A18 respectively, which ends the proof of (8).

Now we return to the proof of (e). Since $\vdash h(\lambda)$, we get $\vdash q(h(\lambda))$ by (5). Hence, by (8) and r_0 we conclude that $\vdash h(p(\lambda))$, so $\vdash \alpha$ which ends the proof of (e).

Let x, y be two different individual variables. Obviously $\rho(x) \rightarrow \rho(y) \notin C_{R_*}(\emptyset)$.

Moreover $C_{R_*}(\{\rho(x) \rightarrow \rho(y)\}) \neq F$. Since in the opposite case $\alpha, \neg\alpha \in C_{R_*}(\{\rho(x) \rightarrow \rho(y)\})$ for every formula α, so $h(\alpha), \neg h(\alpha) \in h(C_{R_*}(\{\rho(x) \rightarrow \rho(y)\}))$ and by (e), we get $\vdash h(\alpha)$ and $\vdash \neg h(\alpha)$, which is impossible. Therefore C_{R_*} is incomplete. ■

According to the Theorem 5 we can introduce the rule of substitution analogously as in AL. We shall use the abbreviations C^*_E, $C^*_\Box$ for the consequence operation C_E and $C_\Box$ strengthened by the substitution rule. The problem of completeness of the extended algorithmic logic strengthened by the substitution rule can be solved in a way similar to the one preserved above.

In the next theorem we shall consider the consequence operations C^*_E and $C^*_\Box$, so we can omit it while reading the paper for the first time.

Theorem 7. *The consequence operations C^*_E and $C^*_\Box$ are incomplete.* □

Proof. A sketch of the proof will be presented. We shall prove this theorem only for the consequence operation C^*_E. First we define the function h analogically to *the Definition 13 but we add the condition* $h(\exists_x \alpha) = h(\alpha)$ *for every* $x \in V$ *and for every generalized formula* α. *The inclusion*

(1) $h(C_E^*(\{\rho(x) \rightarrow \rho(y)\})) \subset C_E(\emptyset)$

is proved similarly as in Theorem 6, though we must add the equality $H_1(\exists_x \alpha) = \exists_{h(g(x))} H_1(\alpha)$ *in the definition of the function* H_1. *Moreover to proof the above inclusion we need two properties:*

(2) For every generalized formula $\alpha \in F_V$*: if* $y \in V \setminus \vartheta(\alpha)$, *then* $h(g(y)) \notin \vartheta(H_1(\alpha))$.
(3) For every program $K \in S$*: if* $y \in V \setminus \vartheta(K)$, *then* $h(g(y)) \notin \vartheta(K')$.

The function q *in this proof is defined for every generalized formula* $\delta \in F_V$ *as follows:*

$$q(\delta) = \begin{cases} h^{e_*}(\delta) & \text{if } \delta \in F_o \\ s^\delta H_1(\delta) & \text{if } \delta \notin F_o \end{cases}$$

The condition (8) in Theorem 6 is checked up for α' *of the form* $\exists_x \lambda$ *in the following way:* $q(h(\alpha')) = q(h([x/\tau]\lambda))$, *while by the inductive hypothesis*

$$\vdash q(h([x/\tau]\lambda)) \leftrightarrow h(p([x/\tau]\lambda)),$$

moreover

$$h(p([x/\tau]\lambda)) = h(s^{\alpha'}([x/\tau]_g^e H(\lambda))),$$

$$\vdash h(s^{\alpha'}([x/\tau]_g^e H(\lambda))) \leftrightarrow h(s^{\alpha'})h(H(\lambda))$$

and

$$h(s^{\alpha'})h(H(\lambda)) = h(s^{\alpha'})h(\exists_{g(x)} H(\lambda)) = h(s^{\alpha'} H(\alpha')) = h(p(\alpha')),$$

thus $\vdash q(h(\alpha')) \leftrightarrow h(p(\alpha'))$.

These remarks enable us to prove (1) and the incompleteness of the extended algorithmic logic with quantifiers strengthened by the substitution rule. ■

4.2 The algorithmic structural completeness of C_{R_*}

In the sequel we shall separate a special class of derivable rules of the consequence operation C_{R_*}. To do that we start with making some remarks about structural and admissible rules. It is easy to see that the rule r_2 is not structural, but instead of it, we can consider a structural rule of the form:

$$\frac{\{\mathscr{S}(\lambda \rightarrow MK^i\alpha) \colon i \in \mathcal{N}\}}{\mathscr{S}(\lambda \rightarrow M \bigcap K\alpha)}$$

where $\mathscr{S}$ denotes any finite sequence of substitutions.

The following remark concerns admissibility of rules. Observe that r_0 is an admissible rule but r_1 is not an admissible one, since for $s = [\alpha/TRUE]$, $K = [\alpha/FALSE]$ where $a \in V_0$ we get

$$<\{a, KTRUE\}, Ka> \in r_1 \text{ and } s\{a, KTRUE\} \subset C_R(\emptyset) \text{ but } s(Ka) \notin C_R(\emptyset).$$

In order to introduce the notion of algorithmic structural completeness we need the special set of generalized formulas $\mathscr{J}$. For example lemma 7 in G. Mirkowska [58] and theorem in G. Mirkowska [59], p. 158, exemplity some forms of the formulas of the set $\mathscr{J}$.

Lemma 7 *For every generalized formula α without symbols $*$, $\bigcap$, $\bigcup$ we can find in an effective way a classical open formula $\alpha_0 \in F_o$ such that for every realization $\mathfrak{R}$ and every valuation $v \in W$, $\alpha_{\mathfrak{R}}(v) = \alpha_{0\mathfrak{R}}(v)$.* □

Theorem 8. *Let K_i, $M_i \in S$, $i \in \{0,1,...,n\}$ be programs in which the sign $*$ does not appear and let $\alpha \in F_o$. Any generalized formula β of the form:*

$$M_0 \bigcup K_0 ... M_n \bigcup K_n \alpha$$

is a tautology of algorithmic logic iff there exists a natural number m such that the formula

$M_0 \bigvee_{i=0}^{m} K_0^i ... M_n \bigvee_{j=0}^{m} K_n^j \alpha$ is a tautology of algorithmic logic where

$M \bigvee_{i=0}^{m} K^i \lambda = M(\lambda \vee K^1 \lambda \vee ... \vee K^m \lambda)$ *for any* $M \in S$, $K \in S$, $\lambda \in F$. □

It is easily seen that for any result of Theorem 8 we can apply Lemma 7 to find a formula $\alpha_0 \in F_o$ which is equivalent to the formula β from Theorem 8.

Since the consequence operation C_{R_*} is incomplete, it accounts for theoretical investigation of algorithmic structural completeness which although weaker, in accordance with our intuition.

Theorem 9. *The consequence operation C_{R_*} of algorithmic logic is algorithmically structurally complete.* □

Proof. Suppose that there exists a structural, finitary and admissible rule r of the consequence operation C_{R_*}, which is not derivable in this consequence operation. Thus there exist a finite set $X \subset \mathscr{J}$ and a generalized formula $\beta \in \mathscr{J}$ such that $<X, \beta> \in r$ and $\beta \notin C_{R_*}(X)$. Let us assume that $X = \{\lambda_1, ..., \lambda_n\}$. According to the definition of the set $\mathscr{J}$ there exist two classical open formulas α, $\lambda \in F_o$ such that $\vdash (\lambda_1 \wedge ... \wedge \lambda_n) \leftrightarrow \lambda$ and $\vdash \beta \leftrightarrow \alpha$. Hence $\alpha \notin C_{R_*}(\{\lambda\})$. By structurality and admissibility of the rule r we get for every program-substitution $p \in Sb$ and every $s \in S_o$:

$$\text{if } s(p(X)) \subset C_{R_*}(\emptyset), \quad \text{then } \vdash s(p(\beta)).$$

Since by Corollary 1 $\vdash p(\lambda_1 \wedge ... \wedge \lambda_n) \leftrightarrow p(\lambda)$ and $\vdash p(\beta) \leftrightarrow p(\alpha)$ for every $p \in Sb$, we conclude by Theorem 3 (ii) that

(1) If $\vdash sp(\lambda)$ then $\vdash sp(\alpha)$ for every $p \in Sb$ and every $s \in S_o$.

Let $g \in G$ be a mapping from Definition 2 such that $g(V \cup V_0) \subset (V \cup V_0) \setminus \vartheta(\alpha \wedge \lambda)$. For further considerations we shall need an endomorphism h^e on F_o such that:

$$e(a) = g(a) \wedge \lambda \quad \text{for every} \quad a \in V_0,$$

$$e(TRUE) = TRUE \quad \text{and} \quad e(FALSE) = FALSE,$$

$$e(\rho(\tau_1, ..., \tau_n)) = \rho(g'(\tau_1), ..., g'(\tau_n)) \wedge \lambda$$

for every elementary formula $\rho(\tau_1, ..., \tau_n) \in E$.

Since $g'(\overline{s\tau}) = \overline{g'(s)g'(\tau)}$ for every $\tau \in T_o$ and for every $s \in S_o$ and since $g'(s)\lambda = \lambda$, we get $e(s\rho(\tau_1, ..., \tau_n)) = g'(s)e(\rho(\tau_1, ..., \tau_n))$ for every $s \in S_o$ and any elementary formula $\rho(\tau_1, ..., \tau_n) \in E$. Thus $e \in \mathscr{E}_g$. Let us take (e, g)-*function* u, $u : At \to F_o$ and a mapping h^u being an extension of u to an endomorphism defined on F_o. By Definition 6 we get an endomorphism H on F. By Definition 7 we get a mapping $p : F \to F$ which is defined by using $g \in G$ and $e \in \mathscr{E}_g$. Hence we get $p \in Sb$.

Moreover let e_0 be an endomorphism on F_o such that:

$$e_0(TRUE) = TRUE \quad \text{and} \quad e_0(FALSE) = FALSE,$$

$$e_0(\delta) = \delta \wedge \neg \delta \quad \text{for every} \quad \delta \in At \setminus \{TRUE, FALSE\}.$$

It can be easily seen that $e_0 \in \mathscr{E}_{g_0}$ for g_0 being an identity function on $V \cup V_0$.

Moreover $\vdash h^{e_0}(\delta) \leftrightarrow TRUE$ or $\vdash h^{e_0}(\delta) \leftrightarrow FALSE$ for every classical open formula $\delta \in F_o$.

Let $Y = \{x_1, ..., x_n, a_1, ..., a_m\}$, then by symbol s_Y we denote the substitution of the form

$$[x_1/g(x_1), ..., x_n/g(x_n), a_1/g(a_1), ..., a_m/g(a_m)].$$

By induction on the length of the classical open formula $\delta \in F_o$ we get

(2) $\vdash h^e(\delta) \leftrightarrow ((\overline{s_Y \delta} \wedge \lambda) \vee (h^{eo}(\delta) \wedge \lambda))$ for every $\delta \in F_o$ and every s_Y such that $\vartheta(\delta) \subset Y$.

First we consider the case $\vdash h^{eo}(\delta) \leftrightarrow FALSE$. Then by (2) we get

$\vdash h^e(\delta) \rightarrow (\overline{s_Y\delta} \wedge \lambda)$, so $\vdash h^e(\delta) \rightarrow (\lambda \rightarrow \overline{s_Y\delta})$ for s_Y defined as in (2).

Let us assume that $\vdash h^{eo}(\delta) \leftrightarrow TRUE$. By (2) we get

(3) $\vdash h^e(\delta) \rightarrow (\lambda \rightarrow \overline{s_Y\delta})$ for every classical open formula $\delta \in F_o$ and for s_Y defined as in (2).

If $\vdash h^{eo}(\lambda) \leftrightarrow TRUE$, then by (2) for such Y that $\vartheta(\alpha \wedge \lambda) = Y$ we get

$$\vdash h^e(\lambda) \leftrightarrow (\overline{s_Y\lambda} \vee \neg\lambda).$$

Using r_1 we conclude that $\vdash s_Y h^e(\lambda) \leftrightarrow (s_Y\overline{s_Y\lambda} \vee \neg\overline{s_Y\lambda})$. Since $\vartheta(\lambda) \subset \{x_1, ..., x_n, a_1, ..., a_m\}$, we get $\vartheta(\overline{s_Y\lambda}) \subset g(V \cup V_0)$. Therefore by the definition of the fuction g it follows that $\vartheta(\overline{s_Y\lambda}) \cap \vartheta(\alpha \wedge \lambda) = \emptyset$. Obviously

$$\{x_1, ..., x_n, a_1, ..., a_m\} = \vartheta(\alpha \wedge \lambda), \text{ so } \{x_1, ..., x_n, a_1, ..., a_m\} \cap \vartheta(\overline{s_Y\lambda}) = \emptyset$$

and moreover $\overline{s_Y\overline{s_Y\lambda}} = \overline{s_Y\lambda}$. By A14 we get

$$\vdash s_Y h^e(\lambda) \leftrightarrow (\overline{s_Y\lambda} \vee \neg\overline{s_Y\lambda}).$$

Since $p(\lambda) = h^e(\lambda)$ from Definition 7 then $\vdash s_Y p(\lambda)$. By (1) we conclude that $\vdash s_Y(p(\alpha))$.

On the ground of (3) and by using r_1 $\vdash s_Y p(\alpha) \rightarrow (s_Y\lambda \rightarrow s_Y\overline{s_Y\alpha})$. By modus ponens rule r_0 we get $\vdash s_Y\lambda \rightarrow s_Y\overline{s_Y\alpha}$. Moreover by A14, A1 and r_0 we get $\vdash s_Y\lambda \rightarrow \overline{s_Y\overline{s_Y\alpha}}$.

Simultaneously $\overline{s_Y\overline{s_Y\alpha}} = \overline{s_Y\alpha}$, so $\vdash s_Y\lambda \rightarrow s_Y\alpha$. Moreover by r_1, r_0 we can observe that $s_Y\alpha \in C_{R_*}(\{\lambda\})$. Using the rule r_1 we obtain that $s_Y^{-1} s_Y\alpha \in C_{R_*}(\{\lambda\})$.

Since $\vartheta(\alpha) \subset \{x_1, \ldots, x_n, a_1, \ldots, a_m\}$, $\{x_1, \ldots, x_n, a_1, \ldots, a_m\} \cap g(V \cup V_0) = \emptyset$ and since $g \in G$ is a one-one mapping, we get $\overline{s_Y^{-1}\, \overline{s_Y \alpha}} = \alpha$. Hence by A14 and r_0 we get $\alpha \in C_{R_*}(\{\lambda\})$, which is impossible.

If $\vdash h^e o(\lambda) \leftrightarrow FALSE$, then by the rule of substitution we get $C_{R_*}(\{\lambda\}) = F$, which is impossible. ■

After defining the standard substitution rule by using the set of program-substitutions Sb_χ we can prove the incompleteness of algorithmic logic strengthened by the substitution rule in the language with generalized terms. For this purpose we need to extend Definition 13 by adding a new condition:

(13) $h(\varphi(\tau_1, \ldots, \tau_n)) = h(\chi(\varphi(\tau_1, \ldots, \tau_n)))$ for every non-classical term such that $\varphi \in \Phi_n$, $n \in N$ and $\tau_1, \ldots, \tau_n \in F'$.

Moreover we can prove (in the same way as Theorem 9) that the consequence operation of algorithmic logic strengthened by the rule of substitution is algorithmically structurally complete in the language with generalized terms.

PART II

Chapter 5

Automated theorem proving

5.1 Axioms and Gentzen's rules of inference

In this chapter we shall describe another system of algorithmic logic. It enables us to formulate some problems connected with a retrieval system. It provides a comprehensive tool in automated theorem proving including programs, procedures and functions. We can get an answer whether some relations defined by programs hold and we can prove functional equations in a dynamic way by looking for a special set of axioms (assumptions) and then adding it to the standard set of axioms. We formulate the **RS**-algorithm which enables us to construct a set of axioms for proving some properties of functions and relations defined by programs. By **RS**-algorithm we get the dynamic process of proving functional equations and we can answer the question whether some relations defined by programs hold. It enables us to solve some problems concerning the correctness of programs. The system can be used for giving an expert appraisement. We shall provide the major structures and a sketch of implementation of the above formal system.

We shall say that s is a *sequent* if it is a pair of sequences of generalized formulas. □

We shall write a sequent s in the form $X \Vdash Y$. The symbol $\alpha \in X$ means that α is an element of the sequence X and the symbol $\alpha \in s$ means that $\alpha \in X$ or $\alpha \in Y$. The set of all sequents will be denoted by Seq.

Let ID be a family of sets of equations of the form $t = \tau$, where t, τ are terms.

Definition 14. *If $X \in ID$ then for any classical terms t, u we shall say that t and u are X equivalent iff one of the following conditions holds:*

(1) there exists a sequence $t_1, ..., t_n$ of classical terms such that t is equal to t_1 and u is equal to t_n and for every $i \in \{1, ..., n-1\}$ either t_i is equal to t_{i+1} or one of the classical open formulas $t_i = t_{i+1}$, $t_{i+1} = t_i$ is in X,
(2) there exist $n \in N$ and n-ary functor φ and a sequence of classical terms $t_1, ..., t_n, u_1, ..., u_n$ such that for every $1 \leq i \leq n$, t_i and u_i are X equivalent and t is equal to $\varphi(t_1, ..., t_n)$ and u is equal to $\varphi(u_1, ..., u_n)$. □

Definition 15. *The sequent s the form $X \Vdash Y$ is called an axiom if and only if the sequent s fulfils one of the following conditions:*

(1) There exists a classical term t such that $t = t$ belongs to Y,
(2) $FALSE \in X$ or $TRUE \in Y$ or $X \cap Y \neq \emptyset$,
(3) There exists $X_1 \subset$ ID such that $X_1 \subset X$ and for some n-ary predicate letter ρ and for some classical terms $t_1, ..., t_n, u_1, ..., u_n$ the following property holds: t_i and u_i are X_1 equivalent and $\rho(t_1, ..., t_n) \in X$ and $\rho(u_1, ..., u_n) \in Y$, for every $1 \leq i \leq n$. □

We shall denote the set of all axioms by $Ax^=$. Now we shall introduce the main tool for proving theorems. Let s be a sequence of elements of the set S_0 i.e. the sequence of elements of the form: **begin** $u_1 := w_1; ...; u_n := w_n$ **end**, for some $n \in N$ such that for $1 \leq i \leq n$ we get: if $u_i \in V$, then $w_i \in T_o$ and if u_i is a propositional variable then $w_i \in F_0$.

Definition 16. *If s is understood as it was defined above i.e. as a sequence of the assignment instructions, then $k(sw)$ means the execution of s on the expression w from $F_o \cup T_o$. In other words we replace all u_i by w_i $(1 \leq i \leq n)$ respectively. Sometimes this operation will be done simultaneously, but in this case we shall say that we count the function k in such a way.*

If $\alpha \in F_o$, $\beta \in F_V$, K, $M \in S$ and at least one of the programs K, M is not an assignment instruction then:

$k(s$ **begin** $K;\ M$ **end** $\beta) = s(K(M\beta))$,
$k(s$ **if** α **then** K **else** $M\beta) = s((\alpha \wedge K\beta) \vee (\neg\alpha \wedge M\beta))$,
$k(s$ **while** α **do** $K\beta) = s(p := TRUE) \bigcup$ **begin** $p := p \wedge \alpha;\ K$ **end** $(p \wedge \neg\alpha \wedge \beta)$,
*where p is the least element of the set $V_0 \setminus \vartheta(s$ **while** α **do** $K\beta)$.* □

If K is a program and $i \in \mathcal{N}$, then $K^o = \mathrm{Id}$ and K^i is a sequence of i-times written the program K.

For any Γ, Q, U being the sets of finite sequences of generalized formulas, $U \subset At$, $U \neq \emptyset$, s being the sequences of elements defined as in Definition 16, $K \in S$, $\delta \in F_V \setminus$ At, $\xi \in$ At, α, $\beta \in F_V$, $x \in V$ we define the schemes of the *rules of inference* as follows:

$(k+)\ \dfrac{\Gamma \Vdash k(sK\alpha), Q}{\Gamma \Vdash Q, sK\alpha}$ $(\text{-}k)\ \dfrac{k(sK\alpha), \Gamma \Vdash Q}{\Gamma, sK\alpha \Vdash Q}$

$(\mathrm{P}+)\ \dfrac{\Gamma \Vdash U, Q, \delta}{\Gamma \Vdash Q, \delta, U}$ $(\text{-P})\ \dfrac{U, \Gamma, \delta \Vdash Q}{\Gamma, \delta, U \Vdash Q}$

$(\mathrm{C}+)\ \dfrac{\Gamma \Vdash s\alpha, Q; \Gamma \Vdash s\beta, Q}{\Gamma \Vdash Q, s(\alpha \wedge \beta)}$ $(\text{-C})\ \dfrac{s\alpha, s\beta, \Gamma \Vdash Q}{\Gamma, s(\alpha \wedge \beta) \Vdash Q}$

$(\mathrm{N}+)\ \dfrac{s\alpha, \Gamma \Vdash Q}{\Gamma \Vdash Q, s\neg\alpha}$ $(\text{-N})\ \dfrac{\Gamma \Vdash s\alpha, Q}{\Gamma, s\neg\alpha \Vdash Q}$

$(\mathrm{I}+)\ \dfrac{s\alpha, \Gamma \Vdash s\beta, Q}{\Gamma \Vdash Q, s(\alpha \rightarrow \beta)}$ $(\text{-I})\ \dfrac{\Gamma \Vdash s\alpha, Q; s\beta, \Gamma \Vdash Q}{\Gamma, s(\alpha \rightarrow \beta) \Vdash Q}$

$(\mathrm{A}+)\ \dfrac{\Gamma \Vdash s\alpha, s\beta, Q}{\Gamma \Vdash Q, s(\alpha \vee \beta)}$ $(\text{-A})\ \dfrac{s\alpha, \Gamma \Vdash Q; s\beta, \Gamma \Vdash Q}{\Gamma, s(\alpha \vee \beta) \Vdash Q}$

$(\bigcup+)\ \dfrac{\Gamma \Vdash s\bigcup K(K\alpha), s\alpha, Q}{\Gamma \Vdash Q, s\bigcup K\alpha}$ $(\text{-}\bigcup)\ \dfrac{\{sK^i\alpha, \Gamma \Vdash Q : i \in \mathcal{N}\}}{\Gamma, s\bigcup K\alpha \Vdash Q}$

$(\bigcap+)\ \dfrac{\{\Gamma \Vdash sK^i\alpha, Q : i \in \mathcal{N}\}}{\Gamma \Vdash Q, s\bigcap K\alpha}$ $(\text{-}\bigcap)\ \dfrac{s\bigcap K(K\alpha), s\alpha, \Gamma \Vdash Q}{\Gamma, s\bigcap K\alpha \Vdash Q}$

$(\mathrm{s}+)\ \dfrac{\Gamma \Vdash k(s\xi), Q}{\Gamma \Vdash Q, s\xi}$ $(\text{-s})\ \dfrac{k(s\xi), \Gamma \Vdash Q}{\Gamma, s\xi \Vdash Q}$

$(\forall+)\ \dfrac{\Gamma \Vdash s((x := y)\alpha), Q}{\Gamma \Vdash Q, s\forall_x\alpha}$

where y is the least element of the set V such that $y \notin \vartheta(\{\Gamma, Q, s\})$.

$(\text{-}\forall) = \bigcup_{t \in T_o} (-\forall)_t$ where for every $t \in T_o$:

$(-\forall)\ \dfrac{s\forall_x\alpha, (y := t)(s((x := y)\alpha)), \Gamma \Vdash Q}{\Gamma, s\forall_x\alpha \Vdash Q}$ and y is the least element of the set $V \setminus \vartheta(s\alpha)$.

Let R_{Seq} be the set of all of the above mentioned rules.

The *deductive system* $<\mathcal{L}, Ax^=, R_{Seq}>$ will be called the system of ALQ with identity. We divide all the rules into two groups: (R+) and (-R).

In the next paragraph we shall try to generalize the language $\mathscr{L}$ on the case of the set of *generalized terms T.*

It is known that for every generalized formula α of the language with generalized terms there exists a classical open formula $\chi(\alpha)$ of the language $\mathscr{L}$ such that $(\alpha \to \chi(\alpha)) \wedge (\chi(\alpha) \to \alpha)$ is a tautology in this language. The definition of the mapping χ was introduced by G. Mirkowska [58]. To get a complete characterization it is necessary to add to the previous rules, two rules of the form:

$$(\chi +)\ \frac{\Gamma \Vdash \chi(\rho(\tau_1, ..., \tau_n)), Q}{\Gamma \Vdash Q, \rho(\tau_1, ..., \tau_n)} \qquad (-\chi)\ \frac{\chi(\rho(\tau_1, ..., \tau_n)), \Gamma \Vdash Q}{\Gamma, \rho(\tau_1, ..., \tau_n) \Vdash Q}$$

5.2 Functions and procedures defined by programs

The idea of defining some properties of functions and relations by programs played an important role in our considerations. It can be found in G. Mirkowska and E. Orłowska [63], G. Mirkowska and A. Salwicki [64], [89], A. Szałas [95]. Some problems of elimination of defined symbols were considered by W. Dańko [22] where the halting problem was considered as well.

Let $\varphi_1, ..., \varphi_p$ and $\rho_1, ..., \rho_r$ be symbols not belonging to the language $\mathscr{L}$. We assume that the functor φ_j is m_j-ary and the predicate letter ρ_i is n_i-ary, for any $j \in \{1, ..., p\}$ and $i \in \{1, ..., r\}$. By $\mathscr{L}^*$ we denote the extension of $\mathscr{L}$ obtained by adding the functors $\varphi_1, ..., \varphi_p$ and the predicates $\rho_1, ..., \rho_r$ to the alphabet of $\mathscr{L}$

Let $K_1, ..., K_r$, $M_1, ..., M_p$ be some programs from $\mathscr{L}$ and let $\alpha_1, ..., \alpha_r \in F_0$ and $t_1, ..., t_p \in T_0$ be some classical open formulas and some classical terms respectively such that:

$$\vartheta(K_i\alpha_i) = \{x_1, ..., x_{n_i}\} \text{ for every } i \in \{1, ..., r\},$$

$$\vartheta(M_j t_j) = \{y_1, ..., y_{m_j}\} \text{ for every } i \in \{1, ..., p\}.$$

Now we introduce the following set of equations and equivalences which will be called the *system of functions and procedures defining the notions* $\varphi_1, ..., \varphi_p$ *and* $\rho_1, ..., \rho_r$:

$$(*)\quad \begin{array}{ll} \varphi_1(y_1, ..., y_{m_1})\, M_1 t_1 & \rho_1(x_1, ..., x_{n_1}) \equiv K_1\alpha_1 \\ \text{- - - - - - - - - -} & \text{- - - - - - - - - - - -} \\ \varphi_1(y_1, ..., y_{m_p})\, M_p t_p & \rho_1(x_1, ..., x_{n_r}) \equiv K_r\alpha_r \end{array}$$

where $\alpha \equiv \beta$ is the generalized formula of the form $(\alpha \to \beta) \wedge (\beta \to \alpha)$.

In the language $\mathscr{L}^*$ the sets T_0, At', F_0', F', $F_\forall'$, S_0', S', Seq' and $(Ax^=)'$ are defined analogously to the sets T_o, At, F_0, F, $F_\forall$, S_0, S, Seq and $Ax^=$. To define the set $R_{Seq'}$ analogously to the set R_{Seq} we change using of the rules (P+) and (-P). The rules (P'+) and (-P') i.e. the rules (P+) and (-P) in the language $\mathscr{L}^*$, can be used even if the classical open formula $\delta \in$At' contains $\varphi_j(t_1, ..., t_{m_j})$ for some $1 \leq j \leq p$ where φ_j is from (*). Obviously we get the set (R'+) and (-R') in the language $\mathscr{L}^*$ instead of (R+) and (-R) respectively.

We extend the set of the rules of inference $R_{Seq'}$. We shall consider (see G. Mirkowska and A. Salwicki [64]) two new rules:

$$(r_{cv}+)\quad \frac{\Gamma \Vdash \beta(\varphi_j(\tau_1, ..., \tau_{m_j})/\textbf{begin } y_1 := \tau_1; ...\ y_{m_j} := \tau_{m_j} \textbf{ end} M_j t_j), Q}{\Gamma \Vdash Q, \beta(\varphi_j(\tau_1, ..., \tau_{m_j}))}$$

$$(r'_{cv}+)\quad \frac{\Gamma \Vdash \beta(\rho_i(\tau_1, ..., \tau_{n_i})/\textbf{begin } x_1 := \tau_1; ...\ x_{n_i} := \tau_{n_i} \textbf{ end} K_i \alpha_i), Q}{\Gamma \Vdash Q, \beta(\rho_i(\tau_1, ..., \tau_{n_i}))}$$

where $x_1, .., x_{n_i}$ do not belong to $\vartheta(\{\tau_1, ..., \tau_{n_i}\})$ for $i \in \{1, ..., r\}$ and $y_1, ..., y_{m_j}$ do not belong to $\varphi_j(\{\tau_1, ..., \tau_{m_j}\})$ for $j \in \{1, ..., p\}$ and $\beta \in$ At'.

Obviously the rules $(-r_{cv})$ and $(-r'_{cv})$ are analogous. Only the assignment instructions from these rules i.e. for example the program **begin** $x_1 := t_1; ...; x_{n_i} := t_{n_i}$ **end** will be executed simultaneously on each classical open formula from At i.e. it will be executed as a function of substitution e:AT $\to T_0'$ such that $e(x_i) = \tau_i$ and $i \in \{1, ..., r\}$.

If s is as in Definition 16 then E(s) $= \{z_1, ..., z_n\}$. After using one of the above mentioned rules we shall need the rule of the form:

$$(B)\quad \frac{\Gamma \Vdash sK\beta(\tau), Q}{\Gamma \Vdash Q, \beta(sK\tau)}$$

where $\beta \in$ At' and every element from $\vartheta(K) \cup$ E(s) is not an element of any term and formula in β except s$K\tau$.

If it is possible, we shall use the rule (B) instead of $(\chi+)$ or $(-\chi)$. Now we define the set W of rules:

$$(r_1+)\quad \frac{\Gamma \Vdash Q(t = t/TRUE)}{\Gamma \Vdash Q} \qquad (r_a)\quad \frac{\Gamma', FALSE, \Gamma'' \Vdash Q}{\Gamma', \varphi_1 = \varphi_2, \Gamma'' \Vdash Q}$$

where φ_1, $\varphi_2 \in \Phi_0$ and in the data structure of integers, the realizations φ_1 and φ_2 are not equal,

$$(r_=+)\quad \frac{\Gamma \Vdash Q(t/\tau)}{\Gamma \Vdash Q}$$

where t is a classical term built from constants, the standard functors $*$, $+$,- (multiplication, addition, substraction) or the functors φ_j from ($*$) for some $1 \leq j \leq p$ and where τ is a classical term, the value of which is equal to the value of the realization of t in the data structure of integers.

Now we describe some rules which change the right side of the symbol $\Vdash$.

($r_{C1}+$) — It replaces all occurrences ($TRUE \wedge \alpha$) or ($\alpha \wedge TRUE$) by α.
($r_{A1}+$) — It replaces all occurrences ($TRUE \vee \alpha$) or ($\alpha \vee TRUE$) by $TRUE$.
($r_{A0}+$) — It replaces all occurrences ($FALSE \vee \alpha$) or ($\alpha \vee FALSE$) by α.
($r_{I1}+$) — It replaces all occurrences ($TRUE \rightarrow \alpha$) by α and ($\alpha \rightarrow TRUE$) by $TRUE$.
($r_{I0}+$) — It replaces all occurrences ($FALSE \rightarrow \alpha$) by $TRUE$ and ($\alpha \rightarrow FALSE$) by $\neg\alpha$.
($r_{C0}+$) — It replaces all occurrences ($FALSE \wedge \alpha$) and ($\alpha \wedge FALSE$) by $FALSE$.
(r_N+) — It replaces all occurrences ($\neg\neg\alpha$) by α.
($r_{N1}+$) — It replaces all occurrences ($\neg TRUE$) by $FALSE$.
($r_{N0}+$) — It replaces all occurrences ($\neg FALSE$) by $TRUE$.

Moreover the analogous rules: ($-r_{C1}$), ($-r_{A1}$), ($-r_{A0}$), ($-r_{I1}$), ($-r_{I0}$), ($-r_{C0}$), ($-r_N$), ($-r_{N0}$), ($-r_{N1}$), ($-r_1$), ($-r_=$) belong to the set W.

By $\mathfrak{R}$ we denote the set containing the rules from W and the rules: ($\chi+$), (-χ), ($r_{cv}+$), (-r_{cv}), (B), (P'+), (-P'), ($r'_{cv}+$), (-r'_{cv}). $\mathfrak{R}$ is the union of two kinds of sets: ($\mathfrak{R}+$) and (-$\mathfrak{R}$).

5.3 Diagram of a formula

In this section we shall consider an extension of the well-known Gentzen's ideas [30], described by G. Mirkowska [58]. At first we recall some auxiliary notions.

The following notions are standard: *tree, root, leaf, level of a tree, height of a tree, path* and *branch.*

If $D = <$ D, $<>$ and D is a tree then by P(x, D) we denote the set of all *immediate successors* of an element x in D. □

Definition 17. <S, < > *is a tree of sequents if and only if* S ⊂ *Seq' and it has exactly one root.* □

If $\mathfrak{S}$ is a tree of sequents <S, < > and s ∈ S is a sequent, then by r(s) we shall understand the rule of the form $\dfrac{P(\mathrm{s}, \mathfrak{S})}{\mathrm{s}}$. It means that s < s' for every sequent s' ∈ P(s, $\mathfrak{S}$).

We shall say that the tree of sequents $<\mathbf{S}, <>$ is *well formed for* $(-\forall')$ if the following property holds:
if the rule $(-\forall')_t$ was used, then earlier we had to use the same rule for each t' ealier than t in the ordered set T_o. □

The set of all generalized formulas on the left side (right side) of the symbol $\Vdash$ of the sequent s will be denoted by left(s) (right(s)).

Definition 18. *A sequent s is called indecomposable in* $\mathcal{L}^*$ *if and only if* $left(s) \cup right(s) \subset At'$, *s contains neither symbol* φ_j *nor* $\rho_i/1 \leq j \leq p$, $1 \leq i \leq r$/*from* (∗) *and if it contains at most classical terms. The other sequents are* decomposable. □

Definition 19. *By a diagram of a sequent* $s \in Seq'$ *with a special set of axioms* $\mathcal{A} \subset Seq'$ *and the rules* $\mathfrak{R}$ *we shall mean the tree of sequents* $\mathfrak{S}_s = <\mathbf{S}, <>$ *if and only if it fulfils the following conditions:*

(1) *The sequent s is a root of* $\mathfrak{S}_s$.
(2) *If* $s' \in \mathbf{S}$ *and s' is indecomposable or if s is an axiom or a special axiom then s' is a leaf.*
(3) *If* $s' \in \mathbf{S}$ *is on the even level of the tree* $\mathfrak{S}$ *and if s' is a conclusion of a rule from X where* $X = (\mathfrak{R}+)\setminus\{r_{cv}+), (r'_{cv}+), (P'+)\}$ *then* $r(s')$ *is an element of X. It means that the order < has the following property: for every sequent s, the expression* $\dfrac{P(s, \mathfrak{S})}{s}$ *is a rule from* $\mathfrak{R}$. *We assume that if s' is decomposable then we consider the first generalized formula on the right side of the considered sequent s' to construct r(s').*
If s' is not a conclusion of a rule from $(\mathfrak{R}+)\setminus\{(r_{cv}+), (r'_{cv}+), (P'+)\}$ *then:*
   (i) *If* $left(s') \cup right(s') \subset At'$ *then the following condition holds:*
      (i1) *If s' is a conclusion of some rule from* $\{(r_{cv}+), (r'_{cv}+), (P'+)\}$ *then* $r(s')$ *is the same element of this set. Otherwise* $r(s') \in (-\mathfrak{R})$. *However if it is one of the rules* $(-r_{cv})$, $(-r'_{cv})$ *or* $(-P')$ *then such* $r(s')$ *is used as the last of all of these rules.*
   (ii) *If* $right(s') \cap ((F' \cup F_\forall)\setminus At') \neq 0$ *then* $r(s') \in (R'+)$.
   (iii) *If* $right(s') \subset At'$ *and* $left(s') \cap ((F' \cup F_\forall)\setminus At') \neq \emptyset$ *then* $r(s') \in (-R')$.
(4) *If* $s' \in \mathbf{S}$ *is on the odd level of the tree* $\mathfrak{S}$ *and if s' is a conclusion of a rule from* $(-\mathfrak{R})\setminus\{(-r_{cv}), (-r'_{cv}), (-P')\}$ *then r(s') is an element from this set. We assume that if s' is decomposable then we consider the first generalized formula on the right side of the considered sequent s' to construct r(s').*
If s' is not a conclusion of a rule from $(-\mathfrak{R})\setminus\{(-r_{cv}), (-r'_{cv}), (-P')\}$ *then:*
   (i) *If* $left(s') \cup right(s') \subset At'$ *then the following condition holds:*
      (i1) *If s' is a conclusion of some rule from* $\{(-r_{cv}), (-r'_{cv}), (-P')\}$ *then* $r(s')$ *is the same element of this set. Otherwise* $r(s') \in (\mathfrak{R}+)$.

However if it is one of the rules (-r_{cu}), (-r'_{cu}) or (-P') then r(s') is used as the last of all of these rules.

(ii) *If left(s') $\cap ((F' \cup F_\forall) \setminus At') \neq \emptyset$ then r(s') $\in$ (-R').*

(iii) *If left(s') $\subset$ At' and right(s') $\cap ((F' \cup F_\forall) \setminus At') \neq \emptyset$ then r(s') $\in$ (R'+).*

(5) $\mathfrak{S}_s$ *is well formed for* (-$\forall$ '). □

The deductive system $<\mathscr{L}^*, (Ax^=)' \cup \mathscr{A}, R_{Seq'} \cup \mathfrak{R}>$ will be called the *retrieval system* where $\mathscr{A}$ is a special set of axioms. □

Using retrieval system we shall change the standard notion of proof which enables us to prove some properties which are not tautologies but which hold in a data structure. Moreover this system enables us to study some notions defined by programs.

We shall say that $\mathfrak{S}_\alpha$ is a *diagram of generalized formula* α if $<S, <>$ is a diagram of the sequent $\emptyset \Vdash \alpha$ with a special set of axioms $\mathscr{A}$ and the rules $\mathfrak{R}$. □

Definition 20. *We shall say that a formula α has the proof in the retrieval system ($\alpha \in proof < (Ax^=)' \cup \mathscr{A}, R_{Seq'} \cup \mathfrak{R} >$) if and only if the height of the diagram of the generalized formula α is finite and each leaf is an axiom or a special axiom.*

However it arises a problem how to choose the set $\mathscr{A}$ of the special axioms. Obviously this problem will be considered in a data structure (in a standard model of arithmetic) in which the functors and predicates are realized and where $+$, $-$, $*$, $()^m$ are interpreted as addition, substraction, multiplication, m-th power respectively.

To explain an algorithm which is looking for the special set of axioms $\mathscr{A}$, first we shall study the example of the function f defined in the introduction:

$$f(n) = \textbf{if } n = 0 \textbf{ then } z := 1 \textbf{ else } z := n*f(n-1); z.$$

We assume that the realization is in the set of integers with the obvious meaning of used symbols.

For further considerations we assume that $Ax_1^= = (Ax^=)' \cup \{s \in Seq' : 1 = u \in right(s)\}$ and $f(n)$ does not contain the individual variable u and u is the least element of the set $V \setminus \vartheta(f(n))$ (we assume that V is well-ordered).

Example 7. *The diagram of the sequent* $\vdash f(1) = u$ *in* $\mathscr{L}^*$ *with a special set of axioms* $\{s \in Seq' : 1 = u \in right(s)\}$ *and the rules* $\mathfrak{R}$ *is finite and each leaf is an axiom in* $\mathscr{L}^*$ i.e. $f(1) = u \in$ proof $<Ax_1^=, R_{Seq'} \cup \mathfrak{R}>$.

Proof. First we construct the diagram of the sequent

(1) $\Vdash f(1) = u$.

Using the rule $(r_{\omega}+)$ and (B+) we get

(2) $\Vdash (n := 1(\text{if } n = 0 \text{ then } z := 1 \text{ else } z := n * f(n-1)\ (z))) = \text{u}$.

By (B) we get

(3) $\Vdash n := 1(\text{if } n = 0 \text{ then } z := 1 \text{ else } z := n * f(n-1)\ (z = \text{u}))$.

By $(k+)$ and (A+) we get

(4) $\Vdash n := 1(n = 0 \wedge (z := 1(z = \text{u}))),\ n := 1(\neg(n = 0) \wedge (z := n * f(n-1)(z = \text{u})))$.

Using (C+) we get two sequents of the form:

(4.1) $\Vdash n := 1(\neg(n = 0)),\ n := 1(n = 0) \wedge (z := 1(z = \text{u}))$),

(4.2) $\Vdash n := 1(z := n * f(n-1)(z = \text{u})),\ n := 1(n = 0 \wedge (z := 1(z = \text{u})))$.

Case (4.1). By (C+) in (4.1) we get two sequents:

(4.1.1) $\Vdash n := 1(n = 0),\ n := 1(\neg(n = 0))$,.

(4.1.2) $\Vdash n := 1(z := 1(z = \text{u})),\ n := 1(\neg(n = 0))$.

Case (4.2). Using (C+) we get two sequents:

(4.2.1) $\Vdash n := 1(n = 0),\ n := 1(z := n * f(n-1)(z = \text{u}))$.

(4.2.2) $\Vdash n := 1(z := 1(z = \text{u})),\ n := 1(z := n * f(n-1)(z = \text{u}))$,

Case (4.1.1). By (N+) we get an axiom.

Case (4.1.2). Using (N+), (-s), (s+) we get $1 = 0 \Vdash 1 = \text{u}$, which by (r_a) is an axiom.

Case (4.2.1). Using (s+), $(r_{=}+)$ and (s+) we get

(5) $\Vdash 1 = 0,\ f(0) = \text{u}$.

By $(r_{\omega}+)$, (P'+) and (B) we get

(6) $\Vdash n := 0(\text{if } n = 0 \text{ then } z := 1 \text{ else } (z := n * f(n-1)(z = \text{u}))),\ 1 = 0$.

By (P'+), $(k+)$, (P'+), (A+) and (P'+) we get

(7) $\Vdash 1 = 0,\ n := 0(n = 0 \wedge (z := 1(z = \text{u}))),\ n := 0 \neg(n = 0) \wedge (z := n * f(n-1)(z = \text{u})))$.

Using (C+) we get two sequents:

(7.1) $\Vdash n := 0(\neg(n = 0)),\ 1 = 0,\ n = 0(n = 0 \wedge (z := 1(z = \text{u})))$.

(7.2) $\Vdash n := 0(z := n * f(n-1)(z = \text{u})),\ 1 = 0,\ n := 0(n = 0) \wedge (z := 1(z = \text{u})))$.

Case (7.1) By (C+) we get two sequents:

(7.1.1) $\Vdash n := 0(n = 0),\ n := 0(\neg(n = 0)),\ 1 = 0$,

(7.1.2) $\Vdash n := 0(z := 1(z = \text{u})),\ n := 0(\neg(n = 0)),\ 1 = 0$.

Case (7.1.1). By (P'+) and (N+) we get

$n := 0(n = 0) \Vdash 1 = 0,\ n := 0(n = 0)$ which is an axiom.

Case (7.1.2). By (P'+), (N+), (+s) and (s+) we get the sequent

$0 = 0 \Vdash 1 = \text{u},\ 1 = 0$, which is a special axiom and which belongs to Ax_1^{-}.

Case (7.2). By (C+) we get two sequents:

(7.2.1) $\Vdash n := 0(n = 0),\ n := 0(z := n * f(n-1)(z = \text{u})),\ 1 = 0$,

(7.2.2) $\Vdash n := 0(z := 1(z = \text{u})),\ n := 0(z := n * f(n-1)(z = \text{u})),\ 1 = 0$.

Case (7.2.1). By (P'+), (s+), $(r_{=}+)$ and (s+) we get

(8) $\Vdash 0 = 0,\ 0 = \text{u},\ 1 = 0$, which by Definition 15 is an axiom in $\mathscr{S}^{*}$ since $0 = 0$ and *TRUE* are equivalent.

Case (7.2.2). Using (P'+) and (s+) we get

(9) $\Vdash (0 * f(0-1)) = \text{u},\ 1 = 0,\ n := 0(z := 1(z = \text{u}))$.

Thus by $(r_{=}+)$ and (s+) we get the sequent $\Vdash 1 = \text{u},\ 0 = \text{u},\ 1 = 0$ from Ax_1^{-}.

Case (4.2.2). Using (s+), $(r_{=}+)$ and (s+) we get the sequent $\Vdash 1 = \text{u},\ f(0) = \text{u}$ from Ax_1^{-}. ∎

It can be seen that the construction of the diagram enables us to find the special set of axioms which are necessary to prove of the above mentioned classical formula $f(1) = \text{u}$. In Lemma 8 we shall explain how to eliminate the case $0 = u, 1 = u$. The interpretation of the functor f and the other functors in a data structure, for example in the set of integers, allows us to choose a special set of axioms to prove the needed properties.

5.4 Retrieval algorithm for functional equations and relations

In this paragraph we shall try to formulate the algorithm which enables us to find a special set of axioms using the premise of function or procedure defining some notions.

Definition 21. *By the premise of function defining the notion* φ_j from $(*)/1 \leq j \leq p/$ *for the classical terms* $t_1, ..., t \in T'_o$ *we mean the classical open formula* $\tau = u$ *for some* $\tau \in T'_o$ *and for* u *being the least individual variable not belonging to the expression defining* φ_j *in* $(*)$. *The formula* $\tau = u$ *enables us to prove* $\varphi_j(t_1, ..., t_m) = u$ *in the language* $\mathscr{L}^*$, *by the special set of axioms* $\{s \in Seq'$: $\tau = u \in right(s)\}$ *and the rules* $R_{Seq'} \cup \mathfrak{R}$.

By the premise of procedure defining ρ_i *from* $(*)/1 \leq i \leq r/$*for the classical terms* $\tau_1, ..., \tau_n \in T'_o$ *we mean either the expression* b, *when we can prove* $\rho_i(\tau_1, ..., \tau_n) \equiv b$ *in the language* $\mathscr{L}^*$, *by the special set of axioms* $\{s \in Seq'$: $b \in right(s)\}$ *and the rules* $R_{Seq'} \cup \mathfrak{R}$ *or the expression* $\neg b$ *when we can prove* $\rho_i(\tau_1, ..., \tau_n) \equiv b$ *by the special set of axioms* $\{s \in Seq'$: $b \in left(s)\}$ *and the same set of rules.*

The premise of function defining the notion φ *will be called the premise of functional equation and the premise of procedure defining* ρ *will be called the premise of relation defined by programs.* □

It can be easily seen that for $\rho_1(x) \equiv p \wedge \neg p$, where $p \in V_0$ and $\tau \in T'_o$ and for $\neg b$ as the only premise we can prove the classical open formula $\rho_1(x) \equiv b$ in the retrieval system by the special set of axioms $\mathscr{A} = \{s \in Seq'$: $b \in left(s)\}$ and the set of rules $R_{Seq'} \cup \mathfrak{R}$. In this case b can be realized as a logical constant FALSE.

We shall give an algorithm which will be able to decide during the execution whether the starting definition of relation ρ_i from $(*)$ $/1 \leq i \leq r/$ is correct. It means that the definition of relation ρ_i is not of the form:

$$\rho_i(x_1, ..., x_{n_i}) \equiv \neg\rho_i(x_1, ..., x_{n_i}).$$

This loop will be eliminated by the following procedure: if during the construction of the proof of $\rho_i(\tau^i_1, ..., \tau^i_{n_i}) \equiv b_i$ in the retrieval system we met b_i and ρ_i on the same side of the symbol $\Vdash$ then STOP — we have to do with the case of the loop in the definition of ρ_i and the proof does not exist.

Example 8.

Let ρ be defined by the following procedure:

$\rho(x) \equiv \neg\rho(x)$.

We shall try to prove $\rho(x) \equiv$ b in the retrieval system, with the empty set of special axioms. Therefore we consider the sequent:

(1) $\Vdash \rho(x) \equiv$ b.

By (C+) and (I+) we get two sequents:

(2) $\rho(x) \Vdash$ b,

(3) b $\Vdash \rho(x)$.

Using the rule (-r_{cv}') to (2) and (r_{cv}'+) to (3) we get

(4) $\neg\rho(x) \Vdash$ b,

(5) b $\Vdash \neg\rho(x)$.

Using (-N) and (N+) for (4) and (5) respectively, we get

(6) $\Vdash \rho(x)$, b,

(7) $\rho(x)$, $b \Vdash$.

If we do not use the above mentioned procedure we shall get the loop, using (r'_{cv}+) to (6), (-r'_{cv}) to (7) and next using (-N) and (N+). ■

The notion of the premise will be explained in the algorithm which will be able to guess for which $\tau \in T'_o$ we shall get $\tau = $ u and whether b or $\neg$b is a premise.

RS-algorithm looking for the premises of functions and procedures defining the notions $\varphi_1, ..., \varphi_p, \rho_1, ..., \rho_r$ of the form:

$$(FP)\ \begin{cases} \varphi_j(y_1,...,y_{m_j}) = M_j t_j \ for\ t^j_1, ..., t^j_{m_j} \in T'_o \\ \rho_j(x_1,...,x_{n_i}) \equiv K_i \alpha_i \ for\ \tau^i_1, ..., \tau^i_{n_i} \in T'_o \end{cases}$$

/$1 \leq j \leq p$ and $1 \leq i \leq r$/ and constructing the special set of axioms in a dynamic process, runs as follows:

(If the main idea is clear to the reader, we suggest omitting the details).

1. j := 1; i := 1;
   Read(k); (k is a natural number helpful for "while")
   $\mathscr{M}$:= an empty file; (It preserves some kind of sequents)
2. n := 0; X := an empty file, which represents the premises of functional equations and relations defined by programs;
   We put the sequent $\Vdash \varphi_j(t^j_1, ..., t^j_m) = u_j$ as the root in the j-diagram and we put the sequent $\Vdash \rho_i(\tau^i_1, ..., \tau^i_n) \equiv b_i$ as the root in the i-diagram, where u_j is the least element of the lineary ordered set $V \backslash \vartheta(\{\varphi_j(t^j_1, ..., t^j_m), Mt_j\})$ such that $u_j \notin \{u_1, ..., u_{j-1}\}$ for j > 1 and where the element b_i is the least element of the lineary ordered set $V_0 \backslash \vartheta(\{\rho_i(\tau^i_1, ..., \tau^i_n), K_i\alpha_i\})$, such that $b_i \notin \{b_1, .., b_{i-1}\}$ for i > 1.
3. If a sequent s on the n-th level is indecomposable or if it is an axiom or a special axiom /i.e. an element of the set $\mathscr{A}$ of the form: $\{s \in \mathrm{Seq}'$: $\alpha \in$ right(s) for some α from X and $\alpha \neq \neg b_m$ for each m $\leq$ i, or $b_m \in$ left(s) for some $\neg b_m \in$ X, m $\leq i \leq r\}$/, then s is a leaf. If s has more than one the same element on the left or right side of the symbol $\Vdash$ then we omit the rest. We check this point after using any rule. If all sequents on the n-th level are leaves then STOP — the proof exists and the set of axioms and special axioms is of the form $(Ax^{=}) \cup \mathscr{A}$.
4. n := n + 1;
   We construct the n-th level of the j-diagram of the sequent $\Vdash \varphi_j(t^j_1, ..., t^j_m) = u_j$ and the n-th level of the i-diagram of the sequent $\Vdash \rho_i(\tau^i_1, ..., \tau^i_n) \equiv b_i$ in $\mathscr{L}^*$ with the rules $\mathfrak{R}$ and the special set of axioms, which was defined above.

If it is possible we use, as in Definition 19, the rule from $(\mathfrak{R} \cup R_{Seq'}) \setminus \{(-\bigcup)\}$ to construct the n-th level in the h-diagram where $h \in \{i,j\}$ by the premises of the considered rule.

If we need use in the construction of the n-th level in the h-diagram the rule $(-\bigcup)$ for a sequent s of the form:
W, $s'(p := TRUE) \bigcup$ **begin** $p := p \wedge \alpha$; K **end** $(p \wedge \neg\alpha \wedge \beta) \Vdash$ Y, b_i, Z, then for further considerations we denote by $M_n(l)$ the expression of the form:
begin s'; $p := TRUE$ **end** [**begin** $p := p \wedge \alpha$; K **end**]^l. We mean that l is a natural number.

It is known that we get the following set of sequents as the result of using the rule $(-\bigcup)$: $\{M_n(l)(p \wedge \neg\alpha \wedge \beta)$, W $\Vdash$ Y, b_i, Z: $l \in \mathcal{N}\}$, but in practice we do not construct all of these elements. We denote Y, b_i, Z by Γ.

Now we consider the following condition for the sequent of the form $M_n(l)(p \wedge \neg\alpha \wedge \beta)$, $W \Vdash \Gamma$: we use RS-algorithm from the point 3 to the sequents: $k(M_n(k)p) \Vdash$; $M_n(k)\neg\alpha \Vdash$; $M_n(k)\beta \Vdash$ and if RS-algorithm gives us the proof of one of the generalized formulas: $\neg k(M_n(k)p)$, $\neg M_n(k)\neg\alpha$, $\neg M_n(k)\beta$, then we assume that the n-th level contains only k-1 elements of the form: $M_n(l)(p \wedge \neg\alpha \wedge \beta)$, $W \Vdash \Gamma$ for $1 \leq l \leq k-1$.
In the opposite case the n-th level contains the sequent

$$s'(p := TRUE) \bigcup \textbf{begin } p := p \wedge \alpha;\ K \textbf{ end } (p \wedge \neg\alpha \wedge \beta),\ W \Vdash \Gamma$$

and additionally it contains either k elements of the form: $M_n(l)(p \wedge \neg\alpha \wedge \beta)$, W $\Vdash \Gamma$ for $1 \leq l \leq k$ when the rule $(-\bigcup)$ is used for the first time for the sequent with regard to $s'(p := TRUE) \bigcup$ **begin** $p := p \wedge \alpha$; K **end** $(p \wedge \neg\alpha \wedge \beta)$ or one element $M_n(k)(p \wedge \neg\alpha \wedge \beta)$, $W \Vdash \Gamma$ when the rule $(-\bigcup)$ is used for the sequent more than once with regard to the above mentioned, generalized formula. (In fact it means that on the n-th level instead of infinite set of sequents $\{M_n(l)(p \wedge \neg\alpha \wedge \beta)$, W $\Vdash \Gamma : l \in \mathcal{N}\}$ we shall consider only a finite number of sequents).

(To have on the n-th level only finite number of sequents we do nearly the same with the rules $(\bigcap +)$ and $(-\forall)$. However instead of the classical term t we put a temporary pointer of dummy d /see P. Gburzyński [29], [28]/. Moreover on each level we have to decide whether some sequents are axioms. To do that we shall use the well-known unification algorithm on the both sides of the sign $\Vdash$).

k := k + 1;

5. We revise the n-th level of the h-diagram and for every sequent s which does not belong to $\mathcal{A} \cup (Ax^{=})'$ we consider two cases:
(i) We look for the classical open formula of the form $\tau = u_j$ in the sequent s such that $\tau = u_j \in$ right(s), τ does not contain the functor φ_j and τ was obtained by none of the rules: $(r_= +)$, $(-r_=)$ applied to a generalized term t containing the functor φ_j and built by the functors: $+, *, /, ()^m$ for some $m \in \mathcal{N}$ /e.g. if in some sequent, the classical term t which is equal to 0 was obtained from $0 * \varphi_j(t_1^j, ..., t_m^j)$ by $(r_= +)$ then the decomposable sequent was changed into the indecomposable sequent and we lost the essential property/.
If we find such a sequent s which fulfils two conditions:
(1) $\alpha \in At'$, for every $\alpha \in s$,
(2) s is not a conclusion of any of the rules from the set $\mathcal{D} = \{(r_1 +), (-r_1), (r_= +), (-r_=), (r_d), (r_\omega +), (-r_\omega), (P'+), (-P'), (B), (r'_\omega +), (-r'_\omega), (\chi +), (-\chi)\}$,
then we consider two cases:

Case 1. If there is another classical open formula of the form $\tau' = u_j$ (we consider this case even if the restriction concerning the rule $(r_= +)$ in the point 5 (i) is not satisfied), then we put this sequent to the file $\mathcal{M}$ unless s is in $\mathcal{M}$. We call this sequent the *special leaf* and we assume that s has no immediate successor.

Hovever, if s is in $\mathscr{M}$ then STOP — if h = j then the proof of the classical open formula $\varphi_j(t_1^j, ..., t_{m_j}^j) = u_j$ in the retrieval system does not exist and if h = i then the proof of the classical open formula $\rho_i(\tau_1^i, ..., \tau_{n_i}^i) \equiv b_i$ in the retrieval system does not exist.

Case 2. If $\tau = u_j$ is the only classical open formula for some τ which fulfils the condition (i), then we put $\tau = u_j$ in the file X and the sequent s becomes a leaf. Then we remove all special leaves from $\mathscr{M}$ containing $\tau = u_j$ on the right of the sign $\Vdash$ and we call them leaves.

(ii) We look for the element b_i in the sequent s. If we find such a sequent containing ρ_i and b_i on the same side of the symbol $\Vdash$ then STOP — we have to do with the case of the form: $\rho_i(x_1, ..., x_{n_i}) \equiv \neg\rho_i(x_1, ..., x_{n_i})$ and the proof does not exist. If s is not a conclusion of any of the rules from $\mathscr{D}$ we consider three cases:

Case 1. If for every $\alpha \in$ left(s) we get $\alpha \in \mathrm{At}' \setminus \{FALSE, b_i\}$ and α does not contain the predicate letter ρ_i and if for every $\beta \in$ right(s) we get $\beta \in \mathrm{At}' \setminus \{TRUE\}$ and β does not contain the predicate letter ρ_i and $b_i \in$ right(s), then we put b_i into the file X.

Case 2. If for every $\alpha \in$ left(s) we get $\alpha \in \mathrm{At}' \setminus \{FALSE\}$ and α does not contain the predicate letter ρ_i and $b_i \in$ left(s) and if for every $\beta \in$ right(s) we get $\beta \in \mathrm{At}' \setminus \{TRUE\}$ and β does not contain the predicate letter ρ_i then we put $\neg b_i$ into the file X.

Case 3. If there is b_i and $\neg b_i$ in X then STOP — the proof of the generalized formula $\rho_i(\tau_1^i, ..., \tau_{n_i}^i) \equiv b_i$ in the retrieval system does not exist.

6. If s is an indecomposable sequent on the n-th level of the h-diagram which is not an element of $\mathscr{A} \cup (\mathrm{Ax}^-)'$ then STOP — the proof of the classical open formula $\varphi_j(t_1^j, ..., t_{m_j}^j) = u_j$ or $\rho_i(\tau_1^i, ..., \tau_{n_i}^i) \equiv b_i$ for the case h = j or h = i respectirely in the retrieval system does not exist.
7. If each indecomposable sequent from the n-th level of the h-diagram is an element of the set $\mathscr{A} \cup (Ax^-)$ and if there is no other sequent on the n-th level then STOP — if h = j then the set $\{s \in \mathrm{Seq}'\colon \alpha \in \mathrm{right}(s)$ for some classical open formula α from X and $\alpha \neq b_d$ and $\alpha \neq \neg b_d$ for $1 \leq d \leq i\}$ is the special set of axioms for the proof of the classical open formulas:

$$\varphi_1(t_1^1, ..., t_{m_1}^1) = u_1, ..., \varphi_j(t_1^j, ..., t_{m_j}^j) = u_j,$$

and the file of the premises of the above functional equations exists and contains all the elements from X which are neither b_d nor $\neg b_d$ for any $d \in \{1, ..., i\}$. However if h = i then the set $\{s \in \mathrm{Seq}'$: $b_d \in$ right(s) for some b_d from the file X where $1 \leq d \leq i$ or $b_d \in$ left(s) for some $\neg b_d$ from the file X where $1 \leq d \leq i\}$ is the special set of axioms for the proof of the generalized formulas:

$$\rho_1(\tau_1^1, ..., \tau_{n_1}^1) \equiv b_1, ..., \rho_i(\tau_1^i, ..., \tau_{n_i}^i) \equiv b_i,$$

and the file of the premises of the above relations defined by programs exists and contains all the elements from X which are of the form b_d or $\neg b_d$ for any $d \in \{1, ..., i\}$.
If $j = p$ and $i = r$ then STOP — $\mathscr{A}$ is the special set of axioms for the proof of all generalized formulas from (FP) and X is the file of the premises of functional equations and relations defined by programs from (FP). If ($j < p$ and $i < r$) or ($j < p$ and $i = r$) or ($i < r$ and $j = p$) then we change i and j respectively and we go to the point 2. □

Now we want to pay attention to a special case, which was mentioned in Case 1 of the point 5 (i). We want to prove in the retrieval system f(2) = u by RS-algorithm.

Example 9.

We start with the sequent:

(1) $\Vdash f(2) = u$.

After using some rules we get among other things two sequents:

(2) $\Vdash (n := 2)(n = 0), (n := 2)(z := n * f(n-1)(z = u))$,

(3) $\Vdash (n := 2)(z := 1\ (z = u)), (n := 2)\ (z := n * f(n-1)(z = u))$.

The continuation of the proof depends on which sequent will be decomposed.

We shall show both of them:

Case 1. If we continue our considerations with the sequent (2), we shall get at last the sequent of the form:

(4) $0 = 0 \Vdash 2 = 0,\ 1 = 0,\ 2 = u$.

By Case 2 of the point 5 (i) of **RS**-algorithm we get $2 = u$ as the premise. Therefore each sequent containing this premise on the left side of the symbol $\Vdash$ is a special axiom. It allows us to end the whole proof.

Case 2. If we continue our proof with the sequent (3) we get at last the sequent of the form:

(5) $\Vdash 0 = u,\ 1 = u,\ 2 = u$.

Moreover 0 in (5) was got from the classical term $2 * (0 * f(0-1))$ by $(r_=+)$. Since we shall not be able to choose only one premise, we call this sequent in the Case 1 of the point 5 (i) of **RS**-algorithm the special leaf and put it into the file $\mathscr{M}$. At that moment we consider other sequents, for example the sequent (2), which allows us to get the premise $2 = u$. By Case 2 of the point 5 (i) of **RS**-algorithm we remove the special leaf from $\mathscr{M}$ and we call it a leaf. It allows us to end the whole proof even in this case. ■

Case 1 in the point 5 (i) is based on the standard model of arithmetic with standard realization.

If there exist only terms without individual variables in the considered programs and formulas, except individual variables of the form x in the expressions $x := \tau$ and if we use only recursive functions and the computations of all programs in (FP) stop then the following lemma holds:

Lemma 8. *Let $\mathfrak{S}_\alpha = \langle \mathbf{S}, < \rangle$ be the diagram of the generalized formula α of the form $\varphi_j(y_1, ..., y_{m_j}) = M_j t_j$ from (FP). If during the execution of* **RS**-*algorithm for the premise of function defining the notion φ_j we get an indecomposable sequent s of the form $\Gamma_1 \Vdash u = \tau_1, ..., u = \tau_n, \Gamma_2$ then there exist two sequents s_1 and s_2 in* **S** *such that $s < s_1$, $s_2 < s_1$, s and s_2 are not compared by $<$ and s_2 contains exactly one classical open formula from $\{u = \tau_1, ..., u = \tau_n\}$ on the right of the symbol $\Vdash$.*

We want to pay attention to one important matter. The diagram $\mathfrak{S}_\alpha$ usually has an infinite path (see the diagram for $f(2) = u$) but using **RS**-algorithm, we get a finite subtree $\langle \mathbf{S}', <' \rangle$ of the tree $\mathfrak{S}_\alpha$ i.e. a finite $\mathbf{S}' \subset \mathbf{S}$ and $<' = <_{/S'}$ where $<_{/S'}$ means the restriction of the relation $<$ to the set S'. Since the computation of the program M_j stops and gives us the result of this computation, this computation points out the path to the sequent s_2. Obviously there is only one sequent s_2 with the above mentioned property.

The main idea of this lemma is the following: if during the execution of **RS**-algorithm we get the path with the indecomposable sequent s of the form $\Gamma_1 \Vdash u = \tau_1, ..., u = \tau_n, \Gamma_2$, (in this case we do not know which τ_j is the calculation of $M_j t_j$), then by the assumption (the computation of all programs in (FP) stops), there exists another path containing the indecomposable sequent s_2 with the only formula of the form $u = \tau_j$ for some $j \in \{1, ..., n\}$ where τ_j is the result of the computation of $M_j t_j$. Therefore during the execution of **RS**-algorithm we stop the calculation along the path with the sequent s and we continue the calculation on the other branch constructed up this moment and we look for the sequent s_2. The sequent s_2 enables us to get the premise of the considered function of the form $u = \tau_j$ and to extend the set of special axioms.

Further we shall give some examples showing that the idea presented in the above algorithm allows us to find the special set of axioms for functions and relations defined by programs.

Using the rules from $R_{Seq'} \cup \mathfrak{R}$ we will be able to find the premises and a special set of axioms to solve the equality of the form $\varphi(t_1, ..., t_m) = u$ for the function defining the notion φ such that $\varphi(t_1, ..., t_m) = Mt$ is from (*) and $u \notin \vartheta(\varphi(t_1, ..., t_m)) \cup \vartheta(Mt)$. Let $\mathscr{A}_i = \{s \in Seq': \alpha \in right(s)$ for some α from the file $X_i\}$ for $i \in \{0, 1, 2, 3\}$ and for the set of premises X_i

Example 10. *There exist the files X_0, X_1, X_2, X_3 of the premises and the special sets of axioms $\mathscr{A}_0$, $\mathscr{A}_1$, $\mathscr{A}_2$, $\mathscr{A}_3$, which are found by using* **RS**-*algorithm during the proof of the following expressions:*

(i) $g(n^4) = u \in proof < (Ax^{-})' \cup \mathscr{A}_0, R_{Seq'} \cup \mathfrak{R} >$,
(ii) $h(1, 2) = u_1 \in proof < (Ax^{-})' \cup \mathscr{A}_1, R_{Seq'} \cup \mathfrak{R} >$,
(iii) $k(x, 1) = u_2 \in proof < (Ax^{-})' \cup \mathscr{A}_2, R_{Seq'} \cup \mathfrak{R} >$,
(iv) $\rho(1, 2) = b \in proof < (Ax^{-})' \cup \mathscr{A}_3, R_{Seq'} \cup \mathfrak{R} >$,

Obviously $\mathscr{A}_0 = \{s \in Seq' : n^4 = u \in right(s)\}$, $\mathscr{A}_1 = \{s \in Seq' : 2 = u_1 \in right(s)\}$, $\mathscr{A}_2 = \{s \in Seq' : x + 1 = u_2 \in right(s)\}$ and $\mathscr{A}_3 = \{s \in Seq' : b \in right(s)\}$.

Proof. (i) To find X_0 an $\mathscr{A}_0$ we make them empty and construct the diagram of sequent
(1) $\Vdash g(n^4) = u$
by **RS**-algorithm in the language $\mathscr{L}^*$ with the special set of rules $\mathfrak{R}$ and the special set of axioms $\mathscr{A}_0$. By $(r_{\omega}+)$ and (B+) we get the sequent
(2) $\Vdash$ **begin** $x := n^4$; $i := n$ **end**(**begin** $i := i + 3$; $z := x$ **end**$(z = u)$).

Thus by (s+) we get
(3) $\Vdash n^4 = u$.

Next by Case 2 of the point 5 of **RS**-algorithm we put $n^4 = u$ into X_0 and therefore the sequent (3) belongs to the set $\mathscr{A}_0$.
(ii) To find the needed X_1 and $\mathscr{A}_1$ we make them empty and construct the diagram of the sequent
(1) $\Vdash h(1, 2) = u_1$
by **RS**-algorithm in the language $\mathscr{L}^*$ with the special set of rules $\mathfrak{R}$ and the special set of axioms $\mathscr{A}_1$. Using $(r_{\omega}+)$ and (B+) we get
(2) $\Vdash$ **begin** $x := 1$; $y := 2$ **end if** $x = 0$ **then** $z := 2$ **else** $z := h(x - 1, h(x, y))$ $(z = u_1)$.

Hence and by (k+), (A+) and (C+) we get two sequents of the form:
(2.1) $\Vdash$ **begin** $x := 1$; $y := 2$ **end**$(\neg(x = 0))$, **begin** $x := 1$; $y := 2$ **end**$((x = 0) \wedge z := 2(z = u_1))$,
(2.2) $\Vdash$ **begin** $x := 1$; $y := 2$ **end** $z := h(x - 1, h(x, y))$ $(z = u_1))$, **begin** $x := 1$; $y := 2$ **end**$((x = 0) \wedge (z := 2(z = u_1)))$.

Case (2.1). Using (C+) we get two sequents of the form:

(2.1.1) $\Vdash$ **begin** $x := 1$; $y := 2$ **end** $(x = 0)$, **begin** $x := 1$; $y := 2$ **end** $(\neg(x = 0))$,

(2.1.2) $\Vdash$ **begin** $x := 1$; $y := 2$ **end** $(z = 2(z = u_1))$, **begin** $x := 1$; $y := 2$ **end** $(\neg(x = 0))$,

Case (2.1.1). By (N+) we get an axiom.

Case (2.1.2). By (N+), (-s), (s+) and (r_a) we get an axiom.

Case (2.2). Using (C+) we get two sequents of the form:

(2.2.1). $\Vdash$ **begin** $x := 1$; $y := 2$ **end** $(x = 0)$, **begin** $x := 1$; $y := 2$ **end** $(z := h(x - 1, h(x, y))$ $(z = u_1))$,

(2.2.2) $\Vdash$ **begin** $x := 1$; $y := 2$ **end** $(z = 2(z = u_1))$, **begin** $x := 1$; $y := 2$ **end** $(z := h(x - 1, h(x, y))$ $(z = u_1))$.

Case (2.2.1). Using the rule (s+), $(r_= +)$ and (s+) we get

(3) $\Vdash 1 = 0, h(0, h(1, 2)) = u_1$.

Using the rule $(r_{cv}+)$, (P'+), (B+), (P'+), $(k+)$ and (P'+) we get

(4) $\Vdash 1 = 0$, **begin** $x := 0$; $y := h(1, 2)$ **end** $(((x = 0) \wedge (z := 2(z = u_1))) \vee (\neg(x = 0) \wedge (z := h(x - 1, h(x, y))(z = u_1))))$.

By (A+) and (P'+) we get

(5) $\Vdash 1 = 0$, **begin** $x := 0$; $y := h(1, 2)$ **end** $((x = 0) \wedge (z := 2(z = u_1)))$, **begin** $x := 0$; $y := h(1, 2)$ **end** $(\neg(x = 0) \wedge (z := h(x - 1, h(x, y))(z = u_1)))$.

For simplicity let us denote by H the second generalized formula on the right-hand side of the above sequent. Let us introduce the following abbreviations:

a = **begin** $x := 0$; $y := h(1, 2)$ **end** $(\neg(x = 0))$,

b = **begin** $x := 0$; $y := h(1, 2)$ **end** $(z := h(x - 1, h(x, y))(z = u_1))$,

c = **begin** $x := 0$; $y := h(1, 2)$ **end** $(x = 0)$,

d = **begin** $x := 0$; $y := h(1, 2)$ **end** $(z := 2(z = u_1))$.

Using in (5) the rule (C+) we get two sequents:

(5.1) $\Vdash a, 1 = 0$, H,

(5.2) $\Vdash b, 1 = 0$, H.

Case (5.1). Using (C+) and (P'+) we get two sequents:

(5.1.1) $\Vdash 1 = 0, c, a$,

(5.1.2) $\Vdash 1 = 0, d, a$.

Case (5.1.1). By (N+) we get an axiom.

Case (5.1.2). By the same rule as used in Case (5.1.1) and by $(-r_1)$ we get

(6) $1 \Vdash 2 = u_1, 1 = 0$. Then by Case 2 of the point 5 of **RS**-algorithm we put $2 = u_1$ into X_1 and therefore the sequent (6) belongs to the set $\mathscr{A}_1$.

Case (5.2). Using (C+) and (P'+) we get two sequents:

(5.2.1) $\Vdash 1 = 0, c, b$,

(5.2.2) $\Vdash 1 = 0, d, b$,

Case (5.2.1). Using twice the rule (s+) and (r_1+) we get an axiom.

Case (5.2.2). Using twice the rule (s+) and $(r_= +)$ we get

(7) $\Vdash 2 = u, h(-1, h(1, 2))) = u_1, 1 = 0$.

The sequent (7) is an element of the set $\mathscr{A}_1$. The case (2.2.2) is similar, so we omit it.

(iii) To find the needed X_2 and $\mathscr{A}_2$ we make them empty and construct the diagram of the sequent $\Vdash k(x, 1) = u_2$ by **RS**-algorithm. The proof is similar to (i). After using many rules we get at last two sequents of the form:

(1) $\Vdash k(x, 0) + 1 = u_2, 1 = 0$,

(1') $\Vdash k(x, 0) + 1 = u_2, x = u_2$.

First we consider (1). By (P'+), $(r_{cv}+)$ and (B+) we get

(2) $\Vdash y := 0$(**if** $y = 0$ **then** $z := x$ **else** $z := k(x, y - 1) + 1$ $(z + 1 = u_2))$, $1 = 0$.

Hence by (P'+), (k+), (A+) and (C+) we get two sequents of the form:

(2.1) $\Vdash y := 0(\neg(y = 0)),\ y := 0(y = 0 \wedge (z := x)(z + 1 = u_2)),\ 1 = 0,$

(2.2) $\Vdash y := 0((z := k(x, y - 1) + 1)(z + 1 = u_2)),\ y := 0(y = 0 \wedge (z := x)(z + 1 = u_2)),\ 1 = 0.$

Case (2.1). Using (P'+), (C+) and (N+) we get an axiom and the sequent of the form: $y := 0(y = 0) \Vdash y := 0((z := x)(z + 1 = u_2)),\ 1 = 0$. After using (+s), (P'+) and (s+) we get $0 = 0 \Vdash z + 1 = u_2, 1 = 0$. Then by Case 2 of the point 5 of **RS**-algorithm we put $z + 1 = u_2$ into X_2 and therefore the sequent (6) belongs to the set $\mathscr{A}_2$.

Case (2.2). By (P'+), (C+), (s+), ($r_=$+) and (r_1+) we get, after some steps, the axiom of the form $\Vdash k(x, -1) + 2 = u_2,\ TRUE,\ 1 = 0$ and the special axiom of the form

$\Vdash k(x, -1) + 2 = u_2,\ x + 1 = u_2,\ 1 = 0.$

The case (1') is similar to the case (1), so we omit it. It is worth to mention, that if we first consider the case (1') we shall get the proof by the Case 1 in the point 5 (i) in **RS**-algorithm.

(iv) To find X_3 and $\mathscr{A}_3$ we make them empty and construct the diagram of the sequent

(1) $\Vdash \rho(1, 2) \equiv b,$

by **RS**-algorithm for $i = 1$, $b_1 = b$ and $k = 2$.

Obviously by (C+) we get two sequents. Next by (I+) we get:

(2) $\rho(1, 2) \Vdash b,$

(3) $b \Vdash \rho(1, 2).$

Using the rule $(-r'_{\omega})$ to (2) and $(r'_{\omega}+)$ to (3) we get

(4) $\textbf{begin } x := 1;\ y := 2 \textbf{ end } K2a \Vdash b,$

(5) $b \Vdash \textbf{begin } x := 1;\ y := 2 \textbf{ end } K2a.$

At first we consider the point (4). By (-k) and (-A) we get two sequents:

(4.1) $\textbf{begin } x := 1;\ y := 2 \textbf{ end } ((x = y) \wedge (a := FALSE)\, a) \Vdash b,$

(4.2) $\textbf{begin } x := 1;\ y := 2 \textbf{ end } (\neg(x = y) \wedge \textbf{begin } u := 0; \textbf{while } \neg((u = y) \vee (u = x)) \textbf{ do } u := u + 1;$ $\textbf{if } u = x \textbf{ then } a := TRUE \textbf{ else } a := FALSE; \textbf{ end } a) \Vdash b.$

Case (4.1). By (-C) and (-s) we get the sequent

$FALSE,\ \textbf{begin } x := 1;\ y := 2 \textbf{ end}(x = y) \Vdash b$ which is an axiom.

Case (4.2). By (-C), (-k) and (-N) we get

(6) $\textbf{begin } x := 1;\ y := 2 \textbf{ end } (u := 0(\textbf{begin while } \neg((u = y) \vee (u = x)) \textbf{ do } u := u + 1; \textbf{ if } u = x \textbf{ then }$ $a := TRUE \textbf{ else } a := FALSE; \textbf{ end } a)) \Vdash \textbf{begin } x := 1;\ y := 2 \textbf{ end } (x = y),\ b.$

By (P'+), (-k), (s+), (-k) and (-r_N) we get

(7) $\textbf{begin } x := 1; y := 2 \textbf{ end}(u := 0\,(p := TRUE \bigcup \textbf{begin } p := (p \wedge \neg((u = y) \vee (u = x))); u := u + 1$ $\textbf{end } (p \wedge ((u = y) \vee (u = x)) \wedge \textbf{if } u = x \textbf{ then } a := TRUE \textbf{ else } a := FALSE\ (a)))) \Vdash 1 = 2, b,$ where p is a special element from V_0 (see Definition 16).

Since we need to use the rule $(-\bigcup)$, by point 4 of **RS**-algorithm we denote by n the level of the considered diagram and by $M_n(l)$ the expression of the form:

$\textbf{begin } x := 1;\ y := 2;\ u := 0;\ p := TRUE \textbf{ end}(\textbf{begin } p := p \wedge \neg((u = y) \vee (u = x)); u := u + 1 \textbf{ end})^l.$

Next we verify whether the sequent $k(M_n(k)p) \Vdash$ has the proof in the retrieval system for $k = 2$.

Since $k(M_n(2)p)$ is of the form $TRUE \wedge \neg((0 = 2) \vee (0 = 1)) \wedge \neg((0 + 1 = 2) \vee (0 + 1 = 1))$, using the rules $(-r_=)$, $(-r_1)$, $(-r_{A1})$, $(-r_{N1})$, $(-r_{C0})$ to the sequent $k(M_n(2)p) \Vdash$ we get the sequent s_0 such that $FALSE \in \mathrm{left}(s_0)$. Hence s_0 is an axiom. By the point 4 of **RS**-algorithm we shall consider only two sequents of the form:

(8) $M_n(l)(p \wedge ((u = y) \vee (u = x)) \wedge (\textbf{if } u = x \textbf{ then } a := TRUE \textbf{ else } a := FALSE; a)) \Vdash 1 = 2, b$ for $l \in \{0, 1\}$.

Using twice (-C) in (8) we get for $l = 1$ the following sequence:

(9) $M_n(1)((u = y) \vee (u = x)), M_n(1)\ \textbf{if }\ u = x\ \textbf{ then }\ a := TRUE\ \textbf{ else }\ a := FALSE\ \ a),\ \ M_n(1)p$ $\Vdash 1 = 2, b.$

Using (-s), (-r_{C1}), (-k), (-A), (-N) and (P'+) we get two sequents:

(9.1) $M_n(1)(u = y), M_n(1)((u = x) \wedge ((a := TRUE)a)) \vee (\neg(u = x) \wedge ((a := FALSE)\ a))) \Vdash 1 = 2, b,$ $(0 = 2) \vee (0 = 1).$

(9.2) $M_n(1)(u = y), M_n(1)((u = x) \wedge ((a := TRUE)a)) \vee (\neg(u = x) \wedge ((a := FALSE)\ a))) \Vdash 1 = 2, b, (0 = 2) \vee (0 = 1)$.

Case (9.1). By (A+), (-A) we get two sequents. Next using (-s) and (r_a) we get axioms, since $FALSE$ is on the left-hand side of the sign $\Vdash$.

Case (9.2). By (A+) and (-A) we get two sequents:

(9.2.1) $M_n(1)((u = y) \wedge ((a := TRUE)a)), M_n(1)(u = x) \Vdash 0 = 2, 0 = 1, 1 = 2), b$,

(9.2.2) $M_n(1)(\neg(u = x) \wedge ((a := FALSE)a)), M_n(1)(u = x) \Vdash 0 = 2, 0 = 1, 1 = 2), b$.

Case (9.2.1) Using (-s), ($-r_=$), (-C) we get

(10) $M_n(1)(u = x), M_n(1)((a := TRUE)a),\ TRUE \Vdash 0 = 2,\ 0 = 1,\ 1 = 2, b$.

By (-P), (-s) and ($-r_=$) we get the sequent of the form:

(11) $TRUE \Vdash 0 = 2,\ 0 = 1,\ 1 = 2,\ b$.

By Case 1 of the point 5 of RS-algorithm we put b into the file X_3 and the sequent from (11) is an element of the set $\mathscr{A}_3$. Moreover the sequent (9.2.2) is the special axiom too. Obviously the sequent (8) for $l = 0$ is the special axiom, since b is in it on the right-hand side of the sign $\Vdash$.

Now we consider the point (5). Using (k+), (A+) and (C+) we get two sequents:

(5.1) $b \Vdash \mathbf{begin}\ x := 1;\ y := 2\ \mathbf{end}\neg(x = y),\ \mathbf{begin}\ x := 1;\ y := 2\ \mathbf{end}((x = y) \wedge (a := FALSE)a)$,

(5.2) $b \Vdash \mathbf{begin}\ x := 0;\ y := 2\ \mathbf{end}(\mathbf{begin}\ u := 1;\ \mathbf{while}\neg((u = y) \vee (u = x))\ \mathbf{do}\ u := u + 1;\ \mathbf{if}\ u = x\ \mathbf{then}\ a := TRUE\ \mathbf{else}\ a := FALSE;\ \mathbf{end}\ a),\ \mathbf{begin}\ x := 1;\ y := 2\ \mathbf{end}\ ((x = y) \wedge ((a := FALSE)a))$.

Case (5.1). By (C+) we get two sequents such that using (N+) for one of them we get an axiom and using (N+), (-P), (s+), (-s) we get the sequent of the form $1 = 2, b \Vdash FALSE$ which by (r_a) becomes an axiom.

Case (5.2). By (C+) we get two sequents:

(5.2.1) $b \Vdash \mathbf{begin}\ x := 1; y := 2\ \mathbf{end}\ (x = y), \mathbf{begin}\ x := 1; y := 2\ \mathbf{end}\ (\mathbf{begin}\ u := 0; \mathbf{while}\neg((u = y) \vee (u = x))\ \mathbf{do}\ u := u + 1;\ \mathbf{if}\ u = x\ \mathbf{then}\ a := TRUE\ \mathbf{else}\ a := FALSE;\ \mathbf{end}\ a)$,

(5.2.2) $b \Vdash \mathbf{begin}\ x := 1;\ y := 2\ \mathbf{end}((a := FALSE)a),\ \mathbf{begin}\ x := 1;\ y := 2\ \mathbf{end}\ (\mathbf{begin}\ u := 0;\ \mathbf{while}\neg((u = y) \vee (u = x))\ \mathbf{do}\ u := u + 1;\ \mathbf{if}\ u = x\ \mathbf{then}\ a := TRUE\ \mathbf{else}\ a := FALSE;\ \mathbf{end}\ a)$.

Since both cases are nearly the same, we shall consider only the case (5.2.1). For further considerations we shall introduce the following abbreviations: α is equal to $((u = y) \vee (u = x))$, $s1$ denotes $\mathbf{begin}\ p := (p \wedge \neg\alpha);\ u := u + 1\ \mathbf{end}$. Using ($k$+), (s+), ($k$+), (P'+) and ($k$+) in (5.2.1) we get

(12) $b \Vdash \mathbf{begin}\ x := 1; y := 2\ \mathbf{end}\ (u := 0\ (p := TRUE \bigcup s1\ (p \wedge \neg\neg\alpha \wedge\ \mathbf{if}\ u = x\ \mathbf{then}\ a := TRUE\ \mathbf{else}\ a := FALSE;\ a))), 1 = 2$, where p is a special element from V_0 (see Definition 16). By (r_N+), (P'+), ($\bigcup$+), (P'+) and (C+) we get two sequents, but one of them, after using ($\bigcup$+), (P+) and (s+), becomes an axiom because $TRUE$ appeares on the right-hand side of the sign $\Vdash$. Therefore we consider only the last sequent which is of the form:

(13) $b \Vdash \mathbf{begin}\ x := 1;\ y := 2\ \mathbf{end}\ (u := 0(p := TRUE\ (\alpha \wedge \mathbf{if}\ u = x\ \mathbf{then}\ a := TRUE\ \mathbf{else}\ a := FALSE; a))), 1 = 2, \mathbf{begin}\ x := 1; y := 2\ \mathbf{end}\ (u := 0\ (p := TRUE \bigcup s1(s1(p \wedge \alpha \wedge\ \mathbf{if}\ u = x\ \mathbf{then}\ a := TRUE\ \mathbf{else}\ a := FALSE;\ a))))$.

Let us denote by δ the sequence of generalized formulas of the form:

$\mathbf{begin}\ x := 1;\ y := 2\ \mathbf{end}\ (u := 0(p := TRUE \bigcup s1(s1(s1(p \wedge \alpha \wedge\ \mathbf{if}\ u = x\ \mathbf{then}\ a := TRUE\ \mathbf{else}\ a := FALSE;\ a))))),\ \mathbf{begin}\ x := 1;\ y := 2\ \mathbf{end}\ (u := 0(p := TRUE(s1(p \wedge \alpha \wedge\ \mathbf{if}\ u = x\ \mathbf{then}\ a := TRUE\ \mathbf{else}\ a := FALSE;\ a))))$.

Using ($\bigcup$+), (P+) and (C+) in (13) we get two sequents:

(13.1) $b \Vdash \mathbf{begin}\ x := 1;\ y := 2\ \mathbf{end}\ (u := 0\ ((p := TRUE)\ \alpha)),\ 1 = 2,\ \delta$,

(13.2) $b \Vdash \mathbf{begin}\ x := 1;\ y := 2\ \mathbf{end}\ (u := 0\ ((p := TRUE)\ \mathbf{if}\ u = x\ \mathbf{then}\ a := TRUE\ \mathbf{else}\ a := FALSE;\ \alpha),\ 1 = 2,\ \delta$.

Let ζ be of the form: $\mathbf{begin}\ x := 1; y := 2\ \mathbf{end}\ (u := 0\ (p := TRUE \bigcup s1(s1(s1(p \wedge \alpha \wedge\ \mathbf{if}\ u = x\ \mathbf{then}\ a := TRUE\ \mathbf{else}\ a := FALSE;\ a)))))$.

Case (13.1). By (C+) we get two sequents:

(13.1.1) $b \Vdash$ begin $x := 1$; $y := 2$ end $(u := 0\ (p := TRUE\ (s1p)))$, begin $x := 1$; $y := 2$ end $(u := 0\ ((p := TRUE)\alpha))$, $1 = 2$, ζ,

(13.1.2) $b \Vdash$ begin $x := 1$; $y := 2$ end $(u := 0\ (p := TRUE\ (s1\ (\alpha \wedge$ if $u = x$ then $a := TRUE$ else $a := FALSE;\ a))))$, begin $x := 1$; $y := 2$ end $(u := 0((p := TRUE\ (\alpha))))$, $1 = 2$, ζ.

Case (13.1.1). By $(\bigcup+)$, (P+), (A+), (s+), $(r_{C1}+)$, (C+) we get two sequents. Using for each of them $(\bigcup+)$, (P+), (s+), (N+), (-A), (r_a) we get four sequents which are axioms because $FALSE$ belongs to the left side of the sign $\Vdash$ of each of them.

Case (13.1.2). Let us consider only the generalized formula Υ of the form: begin $x := 1$; $y := 2$ end $(u := 0(p := TRUE\ (s1\ (\alpha \wedge$ if $u = x$ then $a := TRUE$ else $a := FALSE;\ a))))$, which belongs to the right side of the sign $\Vdash$ in (13.1.2). It is easily seen that using some rules of inference to the sequent (13.1.2) which is of the form $b \Vdash \Upsilon, \Gamma$, we get the sequent of the form:

(14) $b \Vdash \Gamma', \Upsilon$.

Let μ be of the form begin $x := 1$; $y := 2$ end $(u := 0\ (p := TRUE\ (s1((\alpha)))$ and κ be of the form begin $x := 1$; $y := 2$ end $(u := 0(p := TRUE\ (s1($if $u = x$ then $a := TRUE$ else $a := FALSE;\ a))))$. Using (C+) in (14) we get two sequents:

(14.1) $b \Vdash \mu, \Gamma'$,

(14.2) $b \Vdash \kappa, \Gamma'$.

Case (14.1). Using some rules of inference to the generalized formulas which belong to Γ' we get at last some sequents of the form $b \Vdash \Gamma_1, \mu$. Next by (A+) we get sequents of the form:

(15) $b \Vdash \Gamma_2$, begin $x := 1$; $y := 2$ end $(u := 0(p := TRUE\ (s1(u = x), \Gamma_3$.

Repeating this process and using in turn two rules of inference (s+), $(r_= +)$ we get at last some sequents of the form:

$b \Vdash TRUE, \Gamma_4$, which are axioms.

Case (14.2). Using some rules of inference to the generalized formulas from Γ' we get at last some sequents of the form $b \Vdash \Gamma_5, \kappa$. By $(k+)$ we get

(16) $b \Vdash$ begin $x := 1$; $y := 2$ end $(u := 0\ (p := TRUE\ (s1\ (((u = x) \wedge ((a := TRUE)a)) \vee (\neg(u = x) \wedge ((a := FALSE)a)))))$, Γ_5.

Using some rules of inference to the generalized formulas from Γ_5 and at last using (A +) we get the sequent of the form:

(17) $b \Vdash$ begin $x := 1$; $y := 2$ end $(u := 0(p := TRUE\ (s1((u = x) \wedge ((a := TRUE)a))))$, Γ_σ.

Now we use some rules of inference to the generalized formulas from Γ_σ. At last we use (C+) getting two sequents of the form:

(17.1) $b \Vdash$ begin $x := 1$; $y := 2$ end $(u := 0\ (p := TRUE\ (s1\ (u = x))))$, Γ_7,

(17.2) $b \Vdash$ begin $x := 1$; $y := 2$ end $(u := 0\ (p := TRUE\ (s1\ ((a := TRUE)\ a)))$, Γ_7.

Case (17.1). Repeating this process and at last using in turn (s+), $(r_= +)$ and $(r_1 +)$ we get some sequents of the form $b \Vdash TRUE, \Gamma_8$ which are axioms.

Case (17.2). Repeating this process and using at last (s+) we get some sequents of the form $b \Vdash TRUE, \Gamma_7$ which are axioms.

Case (13.2). We shall only show how to use the rule for a special generalized formula of the sequent because the other rules are not essential. Therefore this special generalized formula will be still written on the right side of the considered sequent. By (C+) we get two sequents:

(13.2.1) $b \Vdash \Gamma_9$, begin $x := 1$; $y := 2$ end $(u := 0\ (p := TRUE\ (s1\ p)))$.

(13.2.2) $b \Vdash \Gamma_9$, begin $x := 1$; $y := 2$ end $(u := 0\ (p := TRUE\ (s1(\alpha \wedge$ if $u = x$ then $a := TRUE$ else $a := FALSE;\ a))))$.

It is easily seen that case (13.2.1) is analogous to the case (13.1.1) and the case (13.2.2) is analogous to the case (13.1.2).

We have proved (iv) for $\mathscr{A}_3 = \{s \in Seq' : b \in \mathrm{right}(s)\}$ and X_3 containing the classical formula b. ■

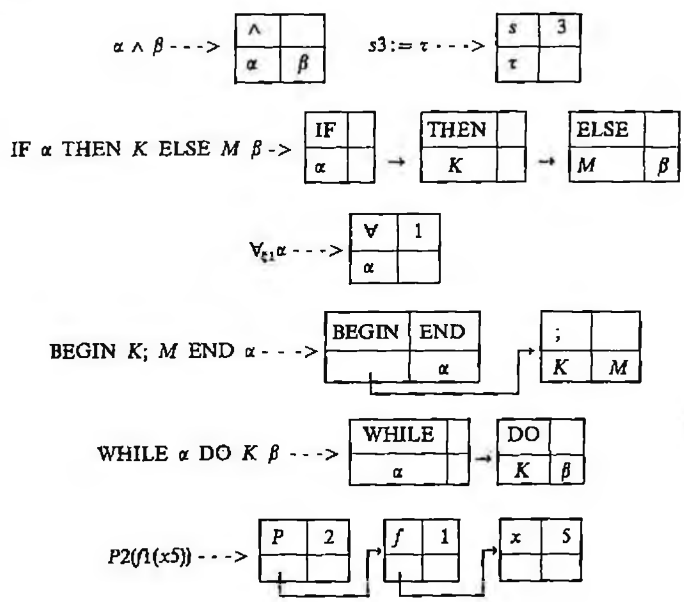
The above examples show that the constructed algorithm computes even such generalized formulas for which the standard computation is helpless, since as it was mentioned in the introduction, it is impossible to compile the program K_5 defining the function $h(x, y)$ in the case $x = 1$ and $y = 2$. The retrieval system, however is able to find the additional premise $u = 2$ of function h to prove the formula $h(1, 2) = u$.

5.5 The data structures and implementation of a retrieval system

The system is based on Gentzen's axiomatization of algorithmic logic G. Mirkowska [58]. The implementation needs some structures. Objects of the type TNODE of the form:

KIND	IDENT
LEFT	RIGHT

where types KIND and IDENT are INTEGER and types LEFT and RIGHT are TNODE represent generalized formulas, generalized terms and programs. We present some representations:



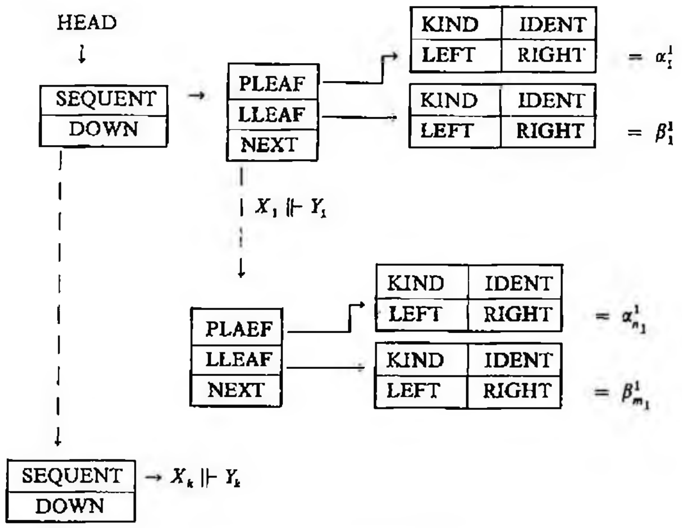
The object of the type FORMULA is of the form:

PLEAF
LLEAF
NEXT

where PLEAF and LLEAF are of the type TNODE and where NEXT is of the type FORMULA. The list of objects of the type FORMULA represents the sequent $X \Vdash Y$. Let POINTER be an object of the form:

SEQUENT
DOWN

where SEQUENT is of the type FORMULA and DOWN is of the type POINTER. Let HEAD be of the type POINTER. We can represent the list of the sequent $X_1 \Vdash Y_1, ..., X_k \Vdash Y_k$ where $X_l \Vdash Y_l = \{\alpha^l_1, ..., \alpha^l_{n_l}\} \Vdash \{\beta^l_1, ..., \beta^l_{m_l}\}$ for $1 \leq l \leq k$ in the following way:



We use the rule only for the last non-empty PLEAF or LLEAF in the considered sequent. It can be seen that (A+) adds a new FORMULA and (C+) generates a new POINTER and a new SEQUENT. Using (C+) to the last non-empty PLEAF in the sequent $X \Vdash Y$, we copy $X \Vdash Y$ and we put a new HEAD1 such that HEAD1. SEQUENT points to the copy of $X \Vdash Y$. Next we pull α from the last non-empty PLEAF in the sequent $X \Vdash Y$. Thus we change TNODE $\alpha \wedge \beta$ into α. Then HEAD1.DOWN:=HEAD.DOWN; HEAD.DOWN:=HEAD1. Moreover we pull β from the last non-empty PLEAF, which lies in the line pointed by HEAD1.SEQUENT. Next we change TNODE $\alpha \wedge \beta$ into β. During the proof we use a lot of options to reduce the complexity of the tree.

In the end we shall provide a sketch of implementation of the retrieval system i.e. we shall present the main procedure PROVE showing only the area of activity of major procedures and functions.

UNIT PROVE : PROCEDURE (M : POINTER);
<Declaration of constants, variables and objects>
BEGIN
Read a formula from a file and construct a sequent pointed by M;
Read a definition of function or relation and construct a sequent pointed by M1;
Replace a function in a sequent by its definition and move a program outside the equality predicate or replace a relation in a sequent by its definition;
While possible, use some basic procedures to the last generalized formula from the right side of M.SEQUENT:
— compute arithmetic expressions /use $(r_{-}+)$/ e.g. $1 + 2 - > 3$,
— compute special arithmetic expressions e.g. $0 * f(x) - > 0$, $x^0 - > 1$,
— convert classical terms $t_1 \neq t_2$ in a model of arithmetic into logical *FALSE* /use (r_a)/,
— convert classical terms, which are equal in the above-mentioned model into logical *TRUE* /use (r_1+)/,
— simplify logical expressions e.g. $TRUE \wedge \alpha - > \alpha$;
Remove a sequent including *FALSE* on left side or *TRUE* on right side;
While a tree M of sequents is not empty, execute the proof:
— while the considered sequent pointed by M contains programs, connectives, functions or predicates defined in M1, continue the proof and look for the set of axioms:
  — if in a considered sequent its antecedent exists then search for a connective or a program in the last formula from right side of M.SEQUENT.
  — If a connective or a program was found, use a proper rule from (R+) or (+R), else either if it is possible make substitutions and move the last formula from the antecedent to the beginning of a sequent, or look for the first formula from the antecedent not belonging to At and move all others formulas on the right of it to the beginning of this sequent,
  — do some ordering procedure simplifying the sequent i.e. remove empty tnodes, search axioms and when found, erase the sequent,
  — repeat the above-mentioned two procedures for the premises of the sequent;
— search for a special axioms which enable us to finish the proof and update the list of them,
— continue the proof for the next sequent after removing the proved sequent.
END.{*PROVE*}

5.6 Results of experiments

Now we discuss some experimental results. In our experiments we use IBM PC/AT with frequency of 50 MHz. Let us consider the following theories:
AR — arithmetic,
AL — algorithmic logic,

ST — set theory,
LT — lattice ; $\mathbf{A}_{LT}$ — axioms of the theory of lattice,
BA — boolean algebra ; $\mathbf{A}_{BA}$ — axioms of the theory of boolean algebra,
G — geometry,
CQ — calculus of quantifiers,
PL — propositional logic.

For further considerations let us define the following program:

K_6 — **if** $x = 1$ **then** $q1 := FALSE$ **else** $q1 := TRUE$;.

Now we define some sets of axioms:

A_1:

a1 — $\forall_x \forall_y (P(x, y) \rightarrow Q(x, y))$
a2 — $\forall_x \forall_y (Q(x, y) \rightarrow R(x, y))$
a3 — $\forall_x \forall_y (R(x, y) \equiv (S(x, y) \vee T(x, y)))$.
a4 — $\forall_x (U(x) \rightarrow W(x))$
a5 — $\forall_x \forall_y (W(x) \equiv \neg(P(y, x) \wedge S(x, y)))$

where

$P(x, y)$ — means that $x \subseteq y$,
$Q(x, y)$ — means that $x \subset y$ or $x = y$,
$R(x, y)$ — means that the power of the set x is less than the power of the set y or these sets are equipollent,
$S(x, y)$ — means that x and y are equipollent,
$T(x, y)$ — means that the power of the set x is less than the power of the set y,
$U(x)$ — means that the set x is finite,
$W(x)$ — means that x is Dedekind finite set;

A_2:

b1 — $\forall_x \forall_y \forall_u \forall_v (T(x, y, u, v) \rightarrow P(x, y, u, v))$
b2 — $\forall_x \forall_y \forall_u \forall_v (P(x, y, u, v) \rightarrow E(x, y, v, u, v, y))$
b3 — $T(a, b, c, d)$

where

$T(x, y, u, v)$ — means that $xyuv$ is a trapezium,
$P(x, y, u, v)$ — means that the segment xy is parallel to uv,
$E(x, y, z, u, v, w)$ — means that the angles xyz and uvw are equal.

IN :

T_1 — $\forall_x (\forall_y (y < x \rightarrow P(y)) \rightarrow P(x)) \rightarrow \forall_x P(x)$
T_2 — $\forall_x \forall_y (x + y = y + x)$
T_3 — $\forall_x (x + c_1 = x)$
T_4 — $\forall_x ((x \neq c_1 \wedge x \neq c_2) \rightarrow c_2 < x)$
T_5 — $\forall_x \forall_y ((x < y) \equiv \exists z (z \neq c_1 \wedge x + z = y)))$
T_6 — $c_1 \neq c_2$

The constants c_1 and c_2 can be interpreted as 0 and 1. This example shows as well that we can use another definition of the mathematical induction.

G_*:

G_1 — $\forall_X \forall_Y \forall_{X'} \forall_{Y'} \forall_\alpha (p_1(X, Y, X', Y', \alpha) \rightarrow \exists_{C1} p_2(X, Y', X', Y, C_1))$
G_2 — $\forall_X \forall_Y \forall_{X'} \forall_{Y'} \forall_\alpha \forall_{C_1} \forall_{C_2} ((p_1(X, Y, X', Y', \alpha) \wedge p_2(X, Y', X', Y, C_1) \wedge P_3(C_1, C_2)) \rightarrow P_4(C_2, \alpha))$
G_3 — $\forall_\alpha \exists_X \exists_Y \exists_{X'} \exists_{Y'} P_1(X, Y, X', Y', \alpha)$
G_4 — $\forall_{C_1} \exists_{C_2} P_3(C_1, C2)$

Moreover the expression $p_1(X, Y, X', Y', \alpha)$ means intuitively that two points X, Y lie on the first arm of the angle and two points X', Y' lie on the second arm of this angle and the pairs of segments OX, OX', XY, X'Y' osculate respectively.

The expression $p_2(X, Y', X', Y, C_1)$ means that the point C_1 lies on two lines XY' and X'Y. The expression $P_3(C_1, C_2)$ means that C_2 is the line OC_1. The expression $P_4(C_2, \alpha)$ means that the line C_2 is a bisectrix of the angle α.

If X is a set of axioms then by $\prod X$ we mean the conjunction of all these axioms. By L we denote the number of used axioms in the considered theory. T denotes the duration of the proof of theorem or the duration of the verification of an expression. We recall that theorems of PL,CQ,AL can be proved without axioms because retrieval system has all necessary rules of inference. By DEF we denote the definition of a function or a relation defined by program (see (FP))

If during the proof of an expression, which should be written in the set DAT, we need a special axiom, then in the column RESULT the premise will be written to inform us about the elements of $\mathscr{A}$.

Table 1

The table of some experimental resules of RS-algorithm

TH	DEF	DAT	L	RESULT	TIME PC 486 50 MHz	
					[m]	[s]
AL	$f(n) = K_1 z$	$f(2) = u$		$\mathscr{A}: u = 2$		0.33
		$f(3) = u$		$\mathscr{A}: u = 6$		0.45
		$f(0) = 1 \wedge (\forall_x(\neg(x = 0) \rightarrow (f(x) = x * f(x-1) \rightarrow f(x+1) = (x+1) * f(x))))$		THEOREM		0.07
	$k(x, y) = K_3(z)$	$k(x, 1) = x + 1$		THEOREM		0.30
		$k(x, 2) = u$		$\mathscr{A}: u = x + 2$		0.32
	$g(x) = K_4(z)$	$g(x^4) = u$		$\mathscr{A}: u = x^4$		0.03
	$\rho(x, y) \equiv K_2 a$	$\rho(1, 2) \equiv b$		$\mathscr{A}: b$ means $b \equiv TRUE$		2.27
	$\rho'(x) \equiv K_6(q1)$	$\rho'(1) \equiv b$		$\mathscr{A}: \neg b$ means $b \equiv FALSE$		0.30
ST		$\prod A1 \rightarrow \forall_x \forall_y((U(x) \wedge P(y, x)) \rightarrow T(y, x))$	5	THEOREM		0.88
LT		$\prod A_{LT} \rightarrow \forall_x((\forall_y x \cup y = y \rightarrow x = 0) \wedge (\forall_y x \cap y = y \rightarrow x = 1))$	15	THEOREM		2.56
BA		$\prod A_{BA} \rightarrow ((X \cup Y) \setminus Z = (X \setminus Z) \cup (Y \setminus Z))$	18	THEOREM		2.8
		$\prod A_{BA} \rightarrow \forall_X \forall_Y (X \subset Y \rightarrow \forall_Z((Z \setminus Y) \subset (Z \setminus X)))$	18	THEOREM		0.07
		$\prod A_{BA} \rightarrow \forall_X \forall_Y ((X \subset Y) \equiv \forall_Z((Y \subset Z) \rightarrow ((Z \setminus X) \cap (Z \setminus Y) = Z \setminus Y)))$	18	THEOREM		0.39

Cont. Tab. 1

G		$\prod \mathbf{A2} \to E(a, b, d, c, d, b)$	3	THEOREM		0.18
AR		$\prod \mathbf{IN} \to ((P(\mathbf{0}) \wedge \forall_x(P(x) \to P(x + \mathbf{1}))) \to \forall_x P(x)$		THEOREM		2.10
CQ		$\forall_x \forall_y P(x, y) \equiv \forall_y \forall_x P(x, y)$		THEOREM		0.12
		$\exists_x(\exists_y P(a, y, x) \wedge \exists_z P(b, z, x)) \vee (\exists_y \neg P(y, y, c) \wedge \exists_z \neg P(z, b, d))$		THEOREM		0.16
		$\forall_x((p_1(x) \to \exists_{x_1} p_2(x_1)) \equiv \exists_{x_2}(p_1(x) \to p_2(x_2)))$		THEOREM		0.06
CQ		$((x = y) \wedge (u = z) \wedge P(x, u)) \to P(y, z)$		THEOREM		0.06
		$(P(x) \to \forall_x Q(x)) \equiv \forall_y(P(x) \to Q(y))$		THEOREM		0.10
		$\neg(\exists_x P(x) \vee \exists_y Q(y)) \vee \exists_z(P(z) \vee Q(v))$		IS NOT A THEOREM		0.03
	$P(x) \equiv \neg P(x)$	$P(x) \equiv b$		DEFINITION INCORRECT		0.02
PL		$(P \to (Q \to S)) \to ((P \to Q) \to (P \to S))$		THEOREM		0.01

Chapter 6

Theorem proving by decomposition

6.1 Axiomatization and decomposition

Let us consider the language $\mathscr{L}''' = \langle L, T_o, T, F_o, F', S_o, S, F'_\forall \rangle$ where $F'_\forall$ is defined as $F_\forall$ and additionally it is based on the language with generalized terms T.

Definition 22. *Let M be a program. We say that*

(i) *the program M is* correctly constructed *if it is not a composition,*

(ii) *the program M is of* normal form *iff one of the following conditions holds:*
   (a) *M is correctly constructed,*
   (b) *$M = [M_1, ..., M_n]$ for some correctly constructed programs $M_1, ..., M_n$ and for $n \geq 2$,*

(iii) *the program M is a* normal assignment *iff M is an assignment instruction or $M = [M_1, ..., M_n]$ where $M_1, ..., M_n$ are assignment instructions and $n \geq 2$.*
   A normal assignment will be denoted by Σ. □

Definition 23. *Let θ be a generalized term or a generalized formula and let $\sigma, \sigma_1, ..., \sigma_n$ be assignment instructions, $n \in N$. We define the execution of $\overline{\Sigma\theta}$ as follows:*

(i) *if $\Sigma = \sigma$ then $\overline{\Sigma\theta}$ is the result of execution of substitution σ for the expression θ,*

(ii) *if $\Sigma = [\sigma_1, ..., \sigma_n]$ then $\overline{\Sigma\theta} := \overline{[\sigma_1, ..., \sigma_{n-1}]\overline{\sigma_n\theta}}$.* □

Definition 24. *Let K, L, M, P be the programs. We denote by symbols ν, * the operations defined as follows:*

(i) $\nu P := P$ *when* P *is correctly constructed,*
$\nu[P, K] := [P, \nu K]$ *when* P *is correctly constructed,*
$\nu[[K, L], M] := \nu[K, \nu[L, M]]$.

(ii) $S^* K := [S, K]$ *when* S *is correctly constructed,*
$[K, L]^* M := [K, L^* M]$. □

Let $\mathcal{M}$ be a model, v -- a valuation, K, L, M -- programs, Σ -- a normal assignment, δ, δ_1 — classical formulas /i.e. formulas without programs and quantifiers/ and let σ be an assignment instruction.

Axioms of decomposition

If U, V are programs, then we put

$$?\delta(U; V) = \begin{cases} U & \text{when } \delta_{\mathcal{R}}(v) = \bigvee_o \\ V & \text{when } \delta_{\mathcal{R}}(v) = \bigwedge_o . \end{cases}$$

We denote by symbol $\prec$ the relation defined as follows:

A1 $[\Sigma, [\sigma, K]] \prec [\Sigma^* \sigma, K]$,
$[\Sigma, \sigma] \prec \Sigma^* \sigma$,

A2 $[\Sigma, [K, L]] \prec [\Sigma, (\nu K)^* L]$ if K is not correctly constructed,

A3 $[\Sigma, [\veebar [\delta KL], M]] \prec ?\overline{\Sigma\delta}([\Sigma, (\nu K)^* M]; [\Sigma, (\nu L)^* M])$,
$[\Sigma, \veebar [\delta KL]] \prec ?\overline{\Sigma\delta}([\Sigma, \nu K]; [\Sigma, \nu L])$,

A4 $[\Sigma, [* [\delta K], M]] \prec ?\overline{\Sigma\delta}([\Sigma, (\nu[K, *[\delta K]])^* M]; [\Sigma, M])$,
$[\Sigma, *[\delta K]] \prec ?\overline{\Sigma\delta}([\Sigma, \nu[K, *[\delta K]]]; \Sigma)$,

$$R_{\text{trans}} : \frac{K \prec L, L \prec M}{K \prec M}.$$

We can see that these axioms give us the rules of decomposition. Operations ν and * defined in Definition 24 prepare the program for decomposition in the case when the program is of the form $[K, M]$ and K is not a normal assignment. To explain Definition 24 and the idea of axioms of decomposition let us consider the following example $[[s_1, s_2], [s_3, s_4]]$ where $s_1, \ldots, s_4$ are assignment instructions.

$[[s_1, s_2], [s_3, s_4]] \prec [[s_1, s_2]^* s_3, s_4]$ by **A1**.
$[[s_1, s_2]^* s_3, s_4] = [[s_1, s_2^* s_3], s_4]$ by Definition 24 (ii).
$[[s_1, s_2^* s_3], s_4] = [[s_1, [s_2, s_3]], s_4]$ by Definition 24 (ii).
$[[s_1, [s_2, s_3]], s_4] \prec [s_1, [s_2, s_3]]^* s_4$ by **A1**.

$[s_1, [s_2, s_3]]^{\bullet}s_4 = [s_1, [s_2, s_3]^{\bullet}s_4] = [s_1, [s_2, s_3^{\bullet}s_4]] = [s_1, [s_2, [s_3, s_4]]] = [s_1, s_2, s_3, s_4]$ by Definition 24 (ii).

The expression $[[s_1, s_2], [s_3, s_4]]$ is not an assignment instruction but using the operations v and $^{\bullet}$ we get a normal assignment of the form $[s_1, s_2, s_3, s_4]$.

Lemma 9. *For any programs K, L, M and for every normal assignment $\Sigma = [\sigma_1, \ldots \sigma_n]$ the following conditions hold:*

(i) $vvK = vK$,

(ii) $len(vK) = len(K)$,

(iii) $(vK)_{\mathscr{R}}(v) = K_{\mathscr{R}}(v)$ *for every valuation v and for every realization $\mathscr{R}$ in a non-empty set U and in the boolean algebra $\mathscr{B}_o$,*

(iv) *for every assignment σ the program $\Sigma^{\bullet}\sigma$ is a normal assignment of the form $[\sigma_1, \ldots, \sigma_n, \sigma]$,*

(v) $v((vK)^{\bullet}L) = v[K, L]$,

(vi) $(vK)^{\bullet}((vL)^{\bullet}M) = (v[K, L])^{\bullet}M$. □

Definition 25. *The length of the decomposition of the expression of the form* $[K, L]$ *is equal to:*

1 — *if there do not exist programs K', L' such that $[K, L] \prec [K', L']$*

$n+1$ — *if there exist programs K', L' such that the length of $[K', L']$ equals n and $[K, L] \prec [K', L']$ is of the form of one of the axioms A1—A4.* □

It is easy to prove the following theorem:

Theorem 10. *If $W_1 \prec W_2$ then $(W_1)_{\mathscr{R}} = (W_2)_{\mathscr{R}}$.* □

Lemma 10. *Let Σ, Σ_1 be the normal assignments and let K, L be the programs. If $[\Sigma, K] \prec \Sigma_1$ then $[\Sigma, (vK)^{\bullet}L] \prec [\Sigma_1, L]$.*

Proof. (Induction on the length of the decomposition of the expression $[\Sigma, K]$). If the length of the decomposition of the expression $[\Sigma, K]$ is equal to 2 then A_1 was used. Because Σ_1 is a normal assignment, then $[\Sigma, K] \prec \Sigma_1$ is of the form $[\Sigma, \sigma] \prec \Sigma^{\bullet}\sigma$. Hence $K = \sigma$, $\Sigma_1 = \Sigma^{\bullet}\sigma$, thus $[\Sigma, (vK)^{\bullet}L] = [\Sigma, [\sigma, L]]$ and $[\Sigma, [\sigma, L]] \prec [\Sigma^{\bullet}\sigma, L]$ and $[\Sigma^{\bullet}\sigma, L] = [\Sigma_1, L]$, which ends the proof of the first step of the induction.

Now we consider all cases of decomposition of $[\Sigma, K]$ by the relation $\prec$. Let us assume inductively that for any K', any normal assignment Σ' for which $[\Sigma', K'] \prec \Sigma_1$ and the length of the decomposition of the expression $[\Sigma', K']$ is less than $[\Sigma, K]$, we have $[\Sigma', (vK')^{\bullet}L] \prec [\Sigma_1, L]$.

If A1 of the form $[\Sigma', [\sigma, K']] \prec [\Sigma'^{\bullet}\sigma, K']$ is used then $K = [\sigma, K']$, $\Sigma = \Sigma'$. Since by Definition 24, we have $[\Sigma, (vK)^{\bullet}L] = [\Sigma, [\sigma, (vK')^{\bullet}L]] \prec [\Sigma'^{\bullet}\sigma,$

$(\nu K')^{\bullet} L]$. Using the induction assumption we get $[\Sigma''\sigma, (\nu K')^{\bullet} L] \prec [\Sigma_1, L]$, thus $[\Sigma, (\nu K)^{\bullet} L] \prec [\Sigma_1, L]$, which ends the proof in this case.

Let K', L' be the programs. If A2 of the form $[\Sigma', [K', L']] \prec [\Sigma', (\nu K')^{\bullet} L']$ is used then $K = [K', L']$, $\Sigma = \Sigma'$. In an analogous way we get $[\Sigma, (\nu((\nu K')^{\bullet} L'))^{\bullet} L] \prec [\Sigma_1, L]$ by the induction assumption. Since by Lemma 9 (v) $\nu((\nu K')^{\bullet} L') = \nu[K', L']$ and $[\Sigma, (\nu[K', L'])^{\bullet} L] \prec [\Sigma_1, L]$, we have $[\Sigma, (\nu K)^{\bullet} L] \prec [\Sigma_1, L]$, which ends the proof in this case. Let K', L', M' be the programs. If A3 of the form $[\Sigma, [\veebar[\delta K' L'], M']] \prec ?\overline{\Sigma\delta}([\Sigma, (\nu K')^{\bullet} M']; [\Sigma, (\nu L')^{\bullet} M'])$ was used and since $?\Sigma\delta([\Sigma, (\nu K')^{\bullet} M']; [\Sigma, (\nu L')^{\bullet} M']) \prec \Sigma_1$ then we get $[\Sigma, (\nu K')^{\bullet} M'] \prec \Sigma_1$, in the case $\mathscr{M} \Vdash \Sigma\delta$ (the other case is similar). Thus by the induction assumption we get $[\Sigma, (\nu((\nu K')^{\bullet} M')) * L] \prec [\Sigma_1, L]$. Since by Lemma 9 (vi) $(\nu((\nu K')^{\bullet} M'))^{\bullet} L = (\nu[K', M'])^{\bullet} L = (\nu K')^{\bullet}((\nu M')^{\bullet} L)$ and $[\Sigma, (\nu K')^{\bullet}(\nu M')^{\bullet} L] \prec [\Sigma_1, L]$, we get $[\Sigma, (\nu[\veebar[\delta K', L'], M'])^{\bullet} L] \prec [\Sigma, [\veebar[\delta K', L'], (\nu M')^{\bullet} L]]$ and $[\Sigma, [\veebar[\delta K', L'], (\nu M')^{\bullet} L]] \prec ?\overline{\Sigma\delta}([\Sigma, (\nu K')^{\bullet}((\nu M')^{\bullet} L)]; [\Sigma, (\nu L')^{\bullet}((\nu M')^{\bullet} L)])$ and moreover $?\overline{\Sigma\delta}([\Sigma, (\nu K')^{\bullet}((\nu M')^{\bullet} L)]; [\Sigma, [(\nu L')^{\bullet}((\nu M')^{\bullet} L)]) = [\Sigma, (\nu K')^{\bullet}((\nu M')^{\bullet} L)]$ and $[\Sigma, (\nu K')^{\bullet}((\nu M')^{\bullet} L)] \prec [\Sigma_1, L]$. R_{trans} enables us to finish this case of the proof.

The proof in the case when A3 is of the form $[\Sigma, \veebar[\delta K', L']] \prec ?\overline{\Sigma\delta}([\Sigma, \nu K']; [\Sigma, \nu L'])$ is similar.

Let A4 be of the form $[\Sigma, [*[\delta K'], M']] \prec ?\overline{\Sigma\delta}([\Sigma, (\nu[K', *[\delta K']])^{\bullet} M']; [\Sigma, M'])$ and let $\mathscr{M} \models \neg ?\Sigma\delta$. Then $[\Sigma, M'] \prec \Sigma_1$ and by the induction assumption we get $[\Sigma, (\nu M')^{\bullet} L] \prec [\Sigma_1, L]$. Thus $[\Sigma, (\nu[*[\delta K'], M'])]^{\bullet}] \prec [\Sigma, [*[\delta, K'], (\nu M')^{\bullet} L]]$ and $[\Sigma, [*[\delta K'], (\nu M')^{\bullet} L]] \prec ?\overline{\Sigma\delta}([\Sigma, (\nu[K', *[\delta K']])^{\bullet}((\nu M')^{\bullet} L)]; [\Sigma, (\nu M')^{\bullet} L])$ and $?\overline{\Sigma\delta}([\Sigma, (\nu[K', *[\delta K']])^{\bullet}((\nu M')^{\bullet} L)]; [\Sigma, (\nu M')^{\bullet} L]) = [\Sigma, (\nu M')^{\bullet} L]$ and finally $[\Sigma, (\nu M')^{\bullet} L] \prec [\Sigma_1, L]$.

Let now $\mathscr{M} \models \Sigma\delta$. Then $[\Sigma, (\nu[K', *[\delta K']])^{\bullet} M'] \prec \Sigma_1$. By the induction assumption of the form $[\Sigma, (\nu((\nu[K', *[\delta, K']])^{\bullet} M'))^{\bullet} L] \prec [\Sigma_1, L]$ and since $\nu((\nu[K', *[\delta K']])^{\bullet} M') = (\nu[K', *[\delta K']])^{\bullet}(\nu M')$ we get $[\Sigma, (\nu[*[\delta K'], M'])^{\bullet} L] \prec [\Sigma, [*[\delta K'], (\nu M')^{\bullet} L]]$ and $[\Sigma, [*[\delta K'], (\nu M')^{\bullet} L]] \prec ?\overline{\Sigma\delta}([\Sigma, (\nu[K', *[\delta K']])^{\bullet}((\nu M')^{\bullet} L]); [\Sigma, (\nu M')^{\bullet} L])$ and $?\Sigma\delta([\Sigma, (\nu[K', *[\delta K']])^{\bullet}((\nu M')^{\bullet} L)]; [\Sigma, (\nu M')^{\bullet} L]) = [\Sigma, (\nu[K', *[\delta K']])^{\bullet}((\nu M')^{\bullet} L)]$ and finally $[\Sigma, (\nu[K', *[\delta K']])^{\bullet}((\nu M')^{\bullet} L)] \prec [\Sigma_1, L]$, which ends the proof in this case.

The proof in the case when A4 is of the form $[\Sigma, *[\delta K]] \prec ?\overline{\Sigma\delta}([\Sigma, \nu[K, *[\delta K]]]; \Sigma)$ is similar. ■

Definition 26. *A normal assignment* Σ *is* well-formed *for the program* K *iff* $\vartheta(\overline{s_i w_k}) \cap \vartheta(\overline{s_j w_l}) = \emptyset$ *for every* s_i, $s_j (i \neq j)$ *from* Σ *and for every* w_k, $w_l \in \vartheta(K)$ *where for any expression* x, $\vartheta(x)$ *denotes the set of all individual and propositional variables occurring in* x. □

Definition 27. *We shall say that the program K has* STOP *property in the model $\mathcal{M}$ iff $K_{\mathfrak{R}}(v) \neq LOOP$ for every valuation v in the model $\mathcal{M}$.* □

By Lemma 10 we can prove the following theorem:

Theorem 11. *Let K be a program and Σ be a normal assignment well-formed for K. If $[\Sigma, K]$ has STOP property in the model $\mathcal{M}$ then there exists a normal assignment Σ^K such that $[\Sigma, K] \prec \Sigma^K$.* □

Proof. (The proof is by the induction of the length of K). The case when K is an assignment instruction is trivial since $[\Sigma, \sigma] \prec \Sigma^{\bullet}\sigma$. Therefore by Lemma 9 (iv) $\Sigma^K = \Sigma^{\bullet}\sigma$ is a normal assignment.

Let K be of the form $[L, M]$. Since $[\Sigma, K]$ has STOP property in the model $\mathcal{M}$, for every valuation v we get:

$$\text{LOOP} \neq [\Sigma, [L, M]]_{\mathfrak{R}}(v) = M_{\mathfrak{R}}(L_{\mathfrak{R}}(\Sigma_{\mathfrak{R}}(v))) = M_{\mathfrak{R}}([\Sigma, L]_{\mathfrak{R}}(v)).$$

Hence $[\Sigma, L]_{\mathfrak{R}}(v) \neq$ LOOP for every v. Thus $[\Sigma, L]$ has STOP property in the considered structure. Since the length of $[\Sigma, L]$ is less then the length of $[\Sigma, K]$, by the induction assumption there exists a normal assignment Σ^L such that $[\Sigma, L] \prec \Sigma^L$. By Lemma 10 we get $[\Sigma,(vL)^{\bullet}M] \prec [\Sigma^L, M]$. By A2 and Theorem 10 $[\Sigma^L, M]$ has STOP property in $\mathcal{M}$. Because Σ^L is well formed for M then there exists Σ^{LM} such that $[\Sigma^L, M] \prec \Sigma^{LM}$ which ends the proof in this case.

Let K be of the form $\veebar[\delta LM]$. By A3 $[\Sigma, \veebar[\delta LM]] \prec ?\overline{\Sigma\delta}([\Sigma, vL];$ $[\Sigma, vM])$. Let us assume that $\mathcal{M} \models \overline{\Sigma\delta}$. Hence $[\Sigma, \veebar[\delta L M]] \prec [\Sigma, vL]$. By Theorem 10 we get that the program $[\Sigma, vL]$ has STOP property. Since Σ is well formed for vL and by the induction assumption there exists a normal assignment Σ^{vL} such that the following relation holds $[\Sigma, \veebar[\delta LM]] \prec \Sigma^{vL}$. The proof of the case $M \models \neg\overline{\Sigma\delta}$ is similar.

Moreover the proof in cases when K is one of the form $[\veebar[\delta LM], N]$, $[*[\delta L], M]$ or $*[\delta L]$ is obviously similar. ■

Corollary 2. *By Theorem 11 we get that every program K having the STOP property in a model $\mathcal{M}$ can be decomposed by the decomposition rules to the normal assignment, which we denote by the symbol $K^{\mathcal{M}}$.* □

6.2 Decomposing proving system

Let $\mathcal{M}$ be a model of arithmetic. For any Γ, Q, U being sets of finite sequences of generalized formulas, $U \subset At$, $U \neq \emptyset$, s being a normal assignment, $K \in S$, $\delta \in F' \backslash At$, $\xi \in At$, $\alpha, \beta \in F'$, $x \in V$ we define the schemes of the *rules of inference* as follows:

$$(\mathrm{P}+)\quad \frac{\Gamma\vdash U, Q, \delta}{\Gamma\vdash Q, \delta, U} \qquad (\text{-P})\quad \frac{U, \Gamma, \delta\vdash Q}{\Gamma, \delta, U\vdash Q}$$

$$(\mathrm{N}+)\quad \frac{s\alpha, \Gamma\vdash Q}{\Gamma\vdash Q, s\neg\alpha} \qquad (\text{-N})\quad \frac{\Gamma\vdash s\alpha, Q}{\Gamma, s\neg\alpha\vdash Q}$$

$$(\mathrm{C}+)\quad \frac{\Gamma\vdash s\alpha, Q; \Gamma\vdash s\beta, Q}{\Gamma\vdash Q, s(\alpha\wedge\beta)} \qquad (\text{-C})\quad \frac{s\alpha, s\beta, \Gamma\vdash Q}{\Gamma, s(\alpha\wedge\beta)\vdash Q}$$

$$(\mathrm{A}+)\quad \frac{\Gamma\vdash s\alpha, s\beta, Q}{\Gamma\vdash Q, s(\alpha\vee\beta)} \qquad (\text{-A})\quad \frac{s\alpha, \Gamma\vdash Q; s\beta, \Gamma\vdash Q}{\Gamma, s(\alpha\vee\beta)\vdash Q}$$

$$(\mathrm{I}+)\quad \frac{s\alpha, \Gamma\vdash s\beta, Q}{\Gamma\vdash Q, s(\alpha\to\beta)} \qquad (\text{-I})\quad \frac{\Gamma\vdash s\alpha, Q; s\beta, \Gamma\vdash Q}{\Gamma, s(\alpha\to\beta)\vdash Q}$$

$$(\bigcap+)\quad \frac{\{\Gamma\vdash sK^i\alpha, Q : i\in\mathcal{N}\}}{\Gamma\vdash Q, s\bigcap K\alpha} \qquad (-\bigcap)\quad \frac{s\bigcap K(K\alpha), s\alpha, \Gamma\vdash Q}{\Gamma, s\bigcap K\alpha\vdash Q}$$

$$(\bigcup+)\quad \frac{\Gamma\vdash s\bigcup K(K\alpha), s\alpha, Q}{\Gamma\vdash Q, s\bigcup K\alpha} \qquad (\text{-}\bigcup)\quad \frac{\{sK^i\alpha, \Gamma\vdash Q : i\in\mathcal{N}\}}{\Gamma, s\bigcup K\alpha\vdash Q}$$

$$(\mathrm{s}+)\quad \frac{\Gamma\vdash(\overline{s\xi}), Q}{\Gamma\vdash Q, s\xi} \qquad (\text{-}s)\quad \frac{(\overline{s\xi}), \Gamma\vdash Q}{\Gamma, s\xi\vdash Q} \qquad (k+)_{\mathcal{M}}\quad \frac{\Gamma\vdash[sK]^{\mathcal{M}}\alpha, Q}{\Gamma\vdash Q, sK\alpha}$$

$$(-k)_{\mathcal{M}}\quad \frac{[sK]^{\mathcal{M}}\alpha, \Gamma\vdash Q}{\Gamma, sK\alpha\vdash Q} \qquad (\forall+)\quad \frac{\Gamma\vdash s((x:=y)\alpha), Q}{\Gamma\vdash Q, s\forall_x\alpha}$$

where y is the least element of the set V such that $y\notin\vartheta(\{\Gamma, Q, s\})$.

$(\text{-}\forall) = \bigcup_{t\in T_o}(-\forall)_t$ where for every $t\in T_o$:

$(-\forall)_t\quad \dfrac{s\forall_x\alpha, (y:=t)(s((x:=y)\alpha)), \Gamma\vdash Q}{\Gamma, s\forall_x\alpha\vdash Q}$ and y is the least element of the set $V\setminus\vartheta(s\alpha)$.

Formulas containing existential quantifiers are transformed in a standard way into equivalent formulas containing universal quantifiers. The rules $(-k)_{\mathcal{M}}$, $(k+)_{\mathcal{M}}$ by Corollary 2 reduce programs to substitutions. Then the rules (s+), (-s) may execute the substitution on atomic formulas.

Let $R^{\mathcal{M}}_{Seq}$ be the set of all of the above mentioned rules and the rules of decomposition. We divide all the rules into two groups: $(R+)^{\mathcal{M}}$ and $(-R)^{\mathcal{M}}$.

It is known by G. Mirkowska [58], [57] and by A. Biela [5] that every generalized formula $\rho(\tau_1, ..., \tau_n)$ containing programs can be transformed by the function χ into the formula of the form $K_1 ... K_m \rho(\tau'_1, ..., \tau'_n)$ where $\rho(\tau'_1, ..., \tau'_n)$ does not contain programs. To get a complete characterization it is necessary to add to the previous rules two rules of the form:

$$(\chi+)\ \frac{\Gamma \vdash \chi(\rho(\tau_1, ..., \tau_n)), Q}{\Gamma \vdash Q, \rho(\tau_1, ..., \tau_n)} \qquad (\text{-}\chi)\ \frac{\chi(\rho(\tau_1, ..., \tau_n)), \Gamma \vdash Q}{\Gamma, \rho(\tau_1, ..., \tau_n) \vdash Q}$$

Let φ and ρ be the symbols not belonging to the considered language. We assume that the functor φ is m-ary and the predicate letter ρ is n-ary. By $\mathcal{L}^*$ we denote the extension obtained by adding the functor φ and the predicate ρ to the alphabet.

Let K, M be programs and let $\alpha \in F_o$ and $t \in T_o$ be such that:

$$\vartheta(K\alpha) = \{x_1, ..., x_n\}$$

$$\vartheta(Mt) = \{y_1, ..., y_m\}$$

Now we introduce in the same way as in chapter 4.2 the *system of function and procedure defining the notions* φ, ρ:

$$(FP) \quad \varphi(y_1, ..., y_{m_1}) = Mt \qquad \rho(x_1, ..., x_{n_1}) \equiv K\alpha$$

In the considered language $\mathcal{L}^*$ the sets At', Seq' and $(Ax^=)'$ are defined analogously to the sets At, Seq and $(Ax^=)$. To define the set $R^{\mathcal{M}}_{Seq'}$ in $\mathcal{L}^*$ analogously to the set $R^{\mathcal{M}}_{Seq}$ we change the usage of the rules (P+) and (-P). The rules (P'+) and (-P') can be used even in the case when the classical formula $\delta \in$ At' and δ contains $\varphi(t_1, ..., t_m)$, where φ is from (FP). Obviously we get the set $(R'+)^{\mathcal{M}}$ and $(-R')^{\mathcal{M}}$ in the language $\mathcal{L}^*$ instead of $(R+)^{\mathcal{M}}$ and $(-R)^{\mathcal{M}}$ respectively.

We extend the set of the rules of inference $R^{\mathcal{M}}_{Seq'}$. We shall consider the following rules:

$$(r_{cv}+),\ (r'_{cv}+),\ (\text{-}r_{cv}),\ (\text{-}r'_{cv}),\ (r_1+),\ (-r_1),\ (r_a),\ (r_=+),\ (-r_=).$$

These rules and the set W are defined analogously as in Chapter 4.2.

We shall need the rule of the form (B) which is one of the rules (B+), (-B) and for example:

$$(\mathrm{B}+)\ \frac{\Gamma \vdash sK(\tau = u), Q}{\Gamma \vdash Q, (sK\tau) = u}$$

where u is not an element of $\vartheta(sK)$. This rule is a case of the rule (B) from Chapter 4.2.

Moreover the rules simplifying generalized formulas containing FALSE or TRUE (e.g. FALSE $\vee \alpha$ changes into α) belong to the set W.

By $\mathcal{R}_*$ we denote the set containing the rules from W and the rules: $(\chi+)$, $(-\chi)$, $(r_{cv}+)$, $(-r_{cv})$, (B), (P'+), (-P'), $(r'_{cv}+)$, $(-r'_{cv})$. $\mathcal{R}_*$ is the union of two sets $(\mathcal{R}_*+)$ and $(-\mathcal{R}_*)$.

6.3 $\mathcal{M}$-diagram

In these considerations the model and the idea of decomposition will be used and the above-mentioned two rules $(k+)_{\mathcal{M}}$, $(-k)_{\mathcal{M}}$ will play an important role.

The notion of indencomposability of a sequent in $\mathcal{L}^*$ is analogous to Definition 18.

Definition 28. *Let $\mathcal{M}$ be a model. By* $\mathcal{M}$-diagram of a sequent $s \in Seq'$ with a special set of axioms $\mathcal{A}_{\mathcal{M}} \subset Seq'$ and the rules $\mathcal{R}_*$ *we shall mean the tree of sequents $\mathfrak{S}_s = \langle S, < \rangle$ if and only if it fulfils the following conditions:*

(1) *The sequent s is a root of $\mathfrak{S}_s$.*
(2) *If $s' \in S$ and s' is indecomposable or s' is an axiom or a special axiom then s' is a leaf.*
(3) *If $s' \in S$ is on the 2n-th level of the tree $\mathfrak{S}$ and s' is a conclusion of a rule from X where $X = (\mathcal{R}_*+) \setminus \{(r_{cv}+), (r'_{cv}+), (P'+)\}$ then $r(s')$ is an element of X. It means that the order $<$ has the following property: for every sequent s, the expression $\frac{P(s, \mathfrak{S})}{s}$ is one of the rules from $\mathcal{R}_*$. We assume that if s' is decomposable then we consider the first generalized formula on the right-hand side of the considered sequent s' to construct $r(s')$.*

   If s' is not a conclusion of a rule from $(\mathcal{R}_*+) \setminus \{(r_{cv}+), (r'_{cv}+), (P'+)\}$ then:
   (i) *If $left(s') \cup right(s') \subset At'$ then the following condition holds:*
      (i1) *If s' is a conclusion of some rule from $\{(r_{cv}+), (r'_{cv}+), (P'+)\}$ then $r(s')$ is the same element of this set. Otherwise $r(s') \in (-\mathcal{R}_*)$. However if it is one of the rules $(-r_{cv})$, $(-r'_{cv})$ or (-P') then such $r(s')$ is used as the last of all of these rules.*
   (ii) *If $right(s') \cap (F'_{\forall} \setminus At') \neq \emptyset$ then $r(s') \in (R'+)^{\mathcal{M}}$.*
   (iii) *If $right(s') \subset At'$ and $left(s') \cap (F'_{\forall} \setminus At') \neq \emptyset$ then $r(s') \in (-R')^{\mathcal{M}}$.*
(4) *If $s' \in S$ is on the 2n + 1-th level of the tree $\mathfrak{S}$ and s' is a conclusion of a rule from $(-\mathcal{R}_*) \setminus \{(-r_{cv}), (-r'_{cv}), (-P')\}$ then $r(s')$ is an element from this set.*

We assume that if s' is decomposable then we consider the first generalized formula on the right-hand side of the considered sequent s' to construct r(s').

If s' is not a conclusion of a rule from $(-\mathcal{R}_*)\setminus\{(-r_{cv}), (-r'_{cv}), (-P')\}$ *then:*

(i) *If* $left(s') \cup right(s') \subset At'$ *then the following condition holds:*

   (i1) *If s' is a conclusion of some rule from* $\{(-r_{cv}), (-r'_{cv}), (-P')\}$ *then r(s') is the same element of this set. Otherwise* $r(s') \in (\mathcal{R}_*+)$. *However if it is one of the rules* $(-r_{cv})$, $(-r'_{cv})$ *or* $(-P')$ *then r(s') is used as the last of all of these rules.*

(ii) *If* $left(s') \cap (F'_\forall \setminus At') \neq \emptyset$ *then* $r(s') \in (-R')^{\mathcal{M}}$.

(iii) *If* $left(s') \subset At'$ *and* $right(s') \cap (F'_\forall \setminus At') \neq \emptyset$ *then* $r(s') \in (R'+)^{\mathcal{M}}$.

(5) $\mathfrak{S}_s$ *is well-formed for* $(-\forall')$. □

The deductive system $<\mathcal{L}^*, (Ax^=)' \cup \mathcal{A}_{\mathcal{M}}, R^{\mathcal{M}}_{Seq'} \cup \mathcal{R}_*>$ will be called the RETRPROV *system* where $\mathcal{A}_{\mathcal{M}}$ is a special set of axioms. □

Using the RETRPROV system we shall change the standard notion of proof in order to prove some properties which are not tautologies but which hold in a model of arithmetic. Moreover this system enables us to consider some notions defined by programs.

We shall say that $\mathfrak{S}_\alpha$ is *$\mathcal{M}$-diagram of generalized formula* α if $<S,<>$ is $\mathcal{M}$-diagram of the sequent $\emptyset \vdash \alpha$ with a special set of axioms $\mathcal{A}_{\mathcal{M}}$ and the rules $\mathcal{R}_*$. □

Definition 29. *We shall say that a generalized formula* α has a proof in the *RETRPROV* system ($\alpha \in proof<(Ax^=)' \cup \mathcal{A}_{\mathcal{M}}, R^{\mathcal{M}}_{Seq'} \cup \mathcal{R}_*>$) *if and only if the height of the $\mathcal{M}$-diagram of the generalized formula α is finite and each leaf is an axiom or a special axiom.* □

However, there is a problem how to choose the set $\mathcal{A}_{\mathcal{M}}$ of the special axioms. Obviously this problem will be considered in a model (i.e. in a standard model of arithmetic in which the functors and predicates are interpreted and where $+$, $-$, $*$, $()^m$,/ are interpreted as addition, subtraction, multiplication, m-th power and division respectively.

Let us remark that the $\mathcal{M}$-diagram of the generalized formula extends Gentzen's ideas: this is shown in Definition 28 (point 3 (i) (i1)), since it makes further proving possible even in the case when we get a sequent containing the atom of the form (FP) or containing the atom including a term of the form (FP) e.g. a term defined by programs (see Example 11).

6.4 Algorithm for proving theorems

In this paragraph we shall formulate the RETRPROV-algorithm which enables us to prove theorems as well as to find a special set of axioms for expressions containing procedures and functions defined by programs.

Let (FP) be a system defining the notions φ and ρ. Any *$\mathcal{M}$-diagram of a generalized formula* $\varphi(t_1, ..., t_m) = u$ from (FP) will be called a φ-$\mathcal{M}$-diagram and any *$\mathcal{M}$-diagram of generalized formula* $\rho(\tau_1, ..., \tau_n) \equiv b$ will be called a ρ-$\mathcal{M}$-diagram $\square$

The **RETRPROV**-algorithm:

1. Read(k); (k is a natural number helpful for "while"),
2. $n := 0$; X:= an empty file. We put the sequent $\vdash \varphi(t_1, ..., t_m) = u$ as the root in the φ-$\mathcal{M}$-diagram (where $\varphi(t_1, ..., t_m)$ is a classical term from (FP)) and we put the sequent $\vdash \rho(\tau_1, ..., \tau_n) \equiv b$ as the root in the ρ-$\mathcal{M}$-diagram (where $\rho(\tau_1, ..., \tau_n)$ is a classical formula from (FP) and $u \in V \setminus \vartheta(\{\varphi(t_1, ..., t_m), Mt\})$. Moreover $b \in V_o \setminus \vartheta(\{\rho(\tau_1, ..., \tau_n), K\alpha\}))$,
3. If a sequent s on the n-th level is indecomposable or if it is an axiom or a special axiom (i.e. it is an element of the set $\mathscr{A}_{\mathcal{M}}$ of the form: $\{s \in \text{Seq'}$: $\alpha \in$right(s) for some α from X and $\alpha \neq \neg b$ or $b \in$left(s) for $\neg b \in$X}), then s is the leaf. If all sequents on the n-th level are leaves then STOP — the proof exists and the set of special axioms is equal to $\mathscr{A}_{\mathcal{M}}$.
4. n := n + 1;
   We construct the n-th level of the φ-$\mathcal{M}$-diagram of the sequent $\vdash \varphi(t_1, ..., t_m) = u$ and the n-th level of the ρ-$\mathcal{M}$-diagram of the sequent $\vdash \rho(_1, ..., \tau_n) \equiv b$ with the rules $\mathscr{R}_*$ and the special set of axioms which was defined above.

   If it is possible we use rules from $(\mathscr{R}_* \cup R^{\mathcal{M}}_{Seq'}) \setminus \{(\text{-}\bigcup)\}$ to construct the n-th level in the φ-$\mathcal{M}$-diagram or in ρ-$\mathcal{M}$-diagram using the premises of the considered rule. We pay attention to use the rules $(r_{cv}+)$, $(-r_{cv})$, (P'+), (-P') only in the case when no other rule from $\mathscr{R}_*$ can be applied.

   If we need use in the construction of the n-th level in the φ-$\mathcal{M}$-diagram or ρ-$\mathcal{M}$-diagram the rule $(\text{-}\bigcup)$ for a sequent s of the form: $\Gamma, s' \bigcup (K\alpha) \vdash Q$, then for further considerations we denote by $M_n(l)$ the expression of the form $s'K^l$ where l is a natural number.

   It is known that we get the following set $\{M_n(l), \Gamma \vdash Q\}$ of sequents as the result of using the rule $(\text{-}\bigcup)$.

   However we do not construct in practice all of these elements. In this case we assume that the n-th level contains the sequent $\Gamma, s' \bigcup (K\alpha) \vdash Q$ and additionally it contains either only $k-1$ elements of the form: $M_n(l), \Gamma \vdash Q$ for $1 \leq l \leq k-1$ when the rule $(\text{-}\bigcup)$ is used for the first time for the sequent $\Gamma, s' \bigcup K\alpha \vdash Q$, or one element $M_n(k), \Gamma \vdash Q$, when the rule $(\text{-}\bigcup)$ was used for the sequent more than once.

   (In fact it means that on the n-th level instead of an infinite set of sequents $\{M_n(l), \Gamma \vdash Q : l \in \mathcal{N}\}$ we shall consider only a finite number of sequents).

(To have on the n-th level only a finite number of sequents we do nearly the same with the rules $(\bigcap +)$ and $(\text{-}\forall)$. However instead of the classical term t we put a temporary pointer of dummy d. Moreover on each level we have to decide whether some sequents are axioms. To do that we shall use the well-known unification algorithm on both sides of the sign $\vdash$).

$k := k + 1$;

5. We revise the n-th level of the φ-$\mathcal{M}$-diagram or ρ-$\mathcal{M}$-diagram. If a sequent $s \in (Ax^{=})'$ then s is a leaf. Otherwise if $s \notin (A^{=})' \cup \mathcal{A}_{\mathcal{M}}$ then we consider two cases:

   (i) We look for a classical formula of the form $\tau = u$ in the sequent s such that $\tau = u \in$ right(s) and τ does not contain the functor φ and τ was not obtained by any of the rules $(r_{=}+)$, $(\text{-}r_{=})$ to a generalized term t containing the functor φ and built by the functors: $+, *, -, /, ()^m$ for some $m \in \mathcal{N}$ (e.g. if in some sequent, the classical term t equal to $\mathbf{0}$ was obtained from $\mathbf{0} * \varphi(t_1, ..., t_m)$ by $(r_{=}+)$, then the decomposable sequent was changed into indecomposable sequent and we have lost the essential property).

   If we find such a sequent s that fulfils two conditions:

   (1) $\alpha \in$ At' for every $\alpha \in s$,

   (2) s is not a conclusion of any of the rules from the set $\mathcal{D} = \{(r_1+), (\text{-}r_1), (r_{=}+), (\text{-}r_{=}), (r_a), (r_{cv}+), (\text{-}r_{cv}), (\text{P}'+), (\text{-P}'), (\text{B}), (r'_{cv}+), (\text{-}r'_{cv}), (\chi+), (\text{-}\chi)\}$, then if $\tau = u$ is the only classical formula for some τ which fulfils the condition (i), we put $\tau = u$ in the file X and the sequent s becomes the leaf.

   (ii) We look for the element b in the sequent s. If we find such a sequent, containing ρ and b on the same side of the symbol $\vdash$ then STOP — we deal with the case of the form: $\rho(x_1, ..., x_n) \equiv \neg\rho(x_1, ..., x_n)$ and the proof does not exist. If s is not a conclusion of any of the rules from $\mathcal{D}$, we consider three cases:

   Case 1. If for every $\alpha \in$ left(s) we get $\alpha \in$ At'$\setminus$ {FALSE, b} and α does not contain the predicate letter ρ and for every $\beta \in$ right(s), $\beta \in$ At'$\setminus$ {TRUE} and β does not contain the predicate letter ρ and $b \in$ *right(s)*, then we put b into the file X.

   Case 2. If for every $\alpha \in$ left(s), $\alpha \in$ At'$\setminus$ {FALSE} and α does not contain the predicate letter ρ and $b \in$ left(s) and for every $\beta \in$ right(s) we get $\beta \in$ At'$\setminus$ {TRUE} and if β does not contain the predicate letter ρ then we put $\neg b$ into the file X.

   Case 3. If there is b and $\neg b$ in X then STOP — the proof of the generalized formula $\rho(\tau_1, ..., \tau_n) \equiv b$ in the RETRPROV system does not exist.

6. If s is an indecomposable sequent on the n-th level of the ρ-$\mathcal{M}$-diagram or φ-$\mathcal{M}$-diagram which is not an element of $\mathcal{A}_{\mathcal{M}} \cup (\text{Ax}^{=})'$ then STOP — if we considered φ-$\mathcal{M}$-diagram then the proof of the classical formula $\varphi(t_1, ..., t_m) = u$ in the RETRPROV system does not exist. Otherwise the proof of the classical formula $\rho(\tau_1, ..., \tau_n) \equiv b$ in the RETRPROV system does not exist.

7. If every indecomposable sequent s from the n-th level of the φ-$\mathcal{M}$-diagram or ρ-$\mathcal{M}$-diagram is an element of the set $\mathcal{A}_{\mathcal{M}} \cup (Ax^{=})$ and if there is no other sequent on the n-th level then STOP — if we considered φ-$\mathcal{M}$-diagram then the set $\{s \in \mathrm{Seq}'\colon \alpha \in \mathrm{right}(s)$ for some classical formula α from X and $\alpha \neq b$ and $\alpha \neq \neg b\}$ is the special set of axioms for the proof of the classical formula:

$$\varphi_1(t_1, \dots, t_m) = u,$$

and the file of the premises of the above mentioned functional equation exists and contains the only element from X which is neither b nor $\neg b$. If we considered ρ-$\mathcal{M}$-diagram then the set $\{s \in \mathrm{Seq}'\colon b \in \mathrm{right}(s)$ for b from the file X or $b \in \mathrm{left}(s)$ for $\neg b$ from the file $X\}$ is the special set of axioms for the proof of the generalized formula:

$$\rho(\tau_1, \dots, \tau_n) \equiv b. \ \square$$

Let us make a remark: the RETRPROV-algorithm can prove the classical formulas as well as the generalized formulas.

6.5 Examples

Now we shall give an example which shows that the idea presented in the above-mentioned algorithm allows us to find a special axiom for a function defined by a program.

To explain the main ideas of the execution of our system let us consider the following example:

Example 11.
$\vdash f(1) = u$ where f is a function defined as follows: $f(n) = K_1 s$. We recall that K_1 means **if** $n = 0$ **then** $s := 1$ **else** $s := n * f(n-1)$.

To prove the above mentioned expression we shall try to find a missing assumption. The execution runs as follows:

$\vdash s_1 := 1($**if** $s_1 = 0$ **then** $s := 1$ **else** $s := s_1 * f(s_1 - 1))(s = u)((r_{cu}+), (B))$,
$\vdash$ **begin** $s_1 := 1;\ s := s_1 * f(s_1 - 1)$ **end** $(s = u)$ .... $((-k)_{\mathcal{M}})$,
(Sometimes for simplicity we shall write such a generalized formula in the form:
$\vdash$ **begin** $s := f(0)$ **end** $(s = u)$.)
$\vdash f(0) = u$ .... $((s+))$,
$\vdash s_2 := 0($**if** $s_2 = 0$ **then** $s_3 := 1$ **else** $s_3 := s_2 * f(s_2 - 1))(s_3 = u)((r_{cu}+), (B))$,
$\vdash$ **begin** $s_2 := 0;\ s_3 := 1$ **end** $(s_3 = u)$ .... $((-k)_{\mathcal{M}})$,
$\vdash 1 = u$ .... $((s+))$.

This system admits the sequent $\vdash 1 = u$ as an additional axiom (a special axiom). Intuitively it means that $f(1) = 1$. So our system proved the expression $\vdash f(1) = u$ by finding an additional axiom and calculated the value of $f(1)$. ■

Example 12.

Let $\vdash f(3) = u$ where $f(n)$ is defined as in Example 11. During the execution of the RETRPROV-algorithm the set $\mathscr{A}_{\mathscr{M}}$ of additional axioms is generated. The proof is identical to the proof in Example 11. In this case $\mathscr{A}_{\mathscr{M}} = \{u = 6\}$. ■

The RETRPROV-algorithm is a dynamic way of looking for a missing axiom necesary to solve a functional equation defined by procedure. Our system can as well find the boolean value of the relation defined by the procedure $P(x) \equiv K\alpha$.

Example 13.

Let us consider the order relation between natural numbers.

(1) $\vdash \rho(1, 2) \equiv b$ where $\rho(x, y) \equiv K_2 q$ and

K_2 means **if** $x_1 = x_2$ **then** q := FALSE **else**

```
begin x3 := 0;
      while ¬((x3 = x2) ∨ (x3 = x1))
          do
            x3 := x3 + 1;
      if x3 = x1
         then q := TRUE else q := FALSE
end
```

(2) $\rho(1, 2) \vdash b$ .... ((C+), (I+)),

(3) $b \vdash \rho(1, 2)$ .... ((C+), (I+)),

(4) **begin** $x_1 := 1$; $x_2 := 2$ **end** $K_2 q \vdash b$ .... ((2) and $(-r'_{cv})$),

(5) $b \vdash$ **begin** $x_1 := 1$; $x_2 := 2$ **end** $K_2 q$ .... ((3) and $(r'_{cv}+)$),

(4.1) **begin** $x_3 := 0$ **while** $\neg((x_3 = 2) \vee (x_3 = 1))$ **do** $x_3 := x_3 + 1$; **if** $x_3 = 1$ **then** $q :=$ TRUE **else** $q :=$ FALSE **end** $q \vdash b$ .... $((-k)_{\mathscr{M}})$,

(4.2) **begin** $x_3 := 0$; $(v($**begin** $x_3 := x_3 + 1$; **while** $\neg((x_3 = 2) \vee (x_3 = 1))$ **do** $x_3 := x_3 + 1$ **end**$))^*$ **if** $x_3 = 1$ **then** $q :=$ TRUE **else** $q :=$ FALSE **end** $q \vdash b$ (by the rule of of decomposition A4),

(4.3) **begin** $x_3 := 0$; (**begin** $x_3 := x_3 + 1$; (**while** $\neg((x_3 = 2) \vee (x_3 = 1))$ **do** $x_3 := x + 1;)^*$ **if** $x_3 = 1$ **then** $q :=$ TRUE **else** $q :=$ FALSE **end**) **end** $q \vdash b$ .... (by Definition 3 (i)),

(4.4) **begin begin** $x_3 := 0$; $x_3 := x_3 + 1$ **end begin** (**while** $\neg((x_3 = 2) \vee (x_3 = 1))$ **do** $x_3 := x_3 + 1$; **if** $x_3 = 1$ **then** $q :=$ TRUE **else** $q :=$ FALSE **end end** $q \vdash b$ .... (by the rule of decomposition A1 and Definition 3 (ii)),

(4.5) **begin begin** $x_3 := 0$; $x_3 := x_3 + 1$ **end if** $x_3 = 1$ **then** $q :=$ TRUE **else** $q :=$ FALSE **end** $q \vdash b$ .... (by the rule of decomposition A4),

(4.6) **begin begin** $x_3 := 0$; $x_3 := x_3 + 1$ **end** $q :=$ TRUE **end** $q \vdash b$ .... (by the rule of decomposition A3 and Definition 2 (i)),

(4.7) **begin** $x_3 := 0$; $x_3 := x_3 + 1$; $q :=$ TRUE **end** $q \vdash b$ .... (by the rule of decomposition A1),

(4.8) TRUE $\vdash b$ .... ((-s)).

By point 5 (case 1) of the RETRPROV-algorithm we put b into X and by point 7 the sequent (4.8) becomes a special axiom in the se $\mathscr{A}_{\mathscr{M}}$, since $\mathscr{A}_{\mathscr{M}} = \{s \in Seq' : b \in \text{right}(s)\}$.

By analogous considerations in the case (5) we get the sequent b $\vdash$ TRUE for $b \vdash \rho(1, 2)$, which is an axiom. ■

Example 14.

Let us consider another example. We consider the function h defined as follows: $h(x_1, x_2) = ($**if** $x_1 = 0$ **then** $x_3 := 2$ **else** $x_3 := h(x_1 - 1, h(x_1, x_2)))x_3$.

We shall try to compute the value of function h(1, 2). Let us remark that some compilers (for example PASCAL, C) cannot do it. However our algorithm manages to solve even this problem, which seems to be rather sophisticated:

(1) $\vdash h(1,2) = u$

(2) $\vdash$ **begin** $x_1 := 1;\ x_2 := 2$ **end** (**if** $x_1 = 0$ **then** $x_3 := 2$ **else** $x_3 := h(x_1 - 1, h(x_1, x_2)))(x_3 = u)$ .... $((r_{cv}+), (B+))$,

(3) $\vdash$ **begin** $x_3 := h(0, h(1,2))$ **end** $(x_3 = u)$ .... $((k+)_{\mathcal{M}}), (r_=+))$,

(4) $\vdash h(0, h(1,2)) = u$ .... $((s+))$,

(5) $\vdash$ **begin** $x_1 := 0;\ x_2 := h(1,2)$ **end**(**if** $x_1 = 0$ **then** $x_3 := 2$ **else** $x_3 := h(x_1 - 1, h(x_1, x_2)))(x_3 = u)$ .... $((r_{cv}+), (B+))$,

(6) $\vdash$ **begin** $x_3 := 2$ **end** $(x_3 = u)$ .... $((k+)_{\mathcal{M}})$,

(7) $\vdash 2 = u$ .... $((s+))$.

By point 5 (i) of the RETRPROV-algorithm we put $u = 2$ into X. By point 7 of this algorithm we get $\mathcal{A}_{\mathcal{M}} = \{s \in Seq' : u = 2 \in \text{right}(s)\}$. So $\vdash u = 2$ becomes a special axiom. ■

Example 15.

Let us consider the function k defined as follows:
$k(x_1, x_2) = ($**if** $x_2 = 0$ **then** $x_3 := x_1$ **else** $x_3 := k(x_1, x_2 - 1) + 1)x_3$.
By using the RETRPROV-algorithm we shall try to prove the expression of the form:

(1) $\vdash k(x, 2) = u$ where x is an individual variable.

(2) $\vdash$ **begin** $x_1 := x;\ x_2 := 2$ **end** (**if** $x_2 = 0$ **then** $x_3 := x_1$ **else** $x_3 := k(x_1, x_2 - 1) + 1)(x_3 = u)$ .... $((r_{cv}+), (B+))$,

(3) $\vdash$ **begin** $x_3 := k(x, 1) + 1$ **end** $(x_3 = u)$ .... $((k+)_{\mathcal{M}})$,

(4) $\vdash k(x, 1) + 1 = u$ .... $((s+))$,

(5) $\vdash$ **begin** $x_1 := x;\ x_2 := 1$ **end** (**if** $x_2 = 0$ **then** $x_3 := x_1$ **else** $x_3 := k(x_1, x_2 - 1) + 1)$ $(x_3 + 1 = u)$ .... $((r_{cv}+), (B+))$,

(6) $\vdash$ **begin** $x_3 := k(x, 0) + 1$ **end** $(x_3 + 1 = u)$ .... $((k+)_{\mathcal{M}})$,

(7) $\vdash k(x, 0) + 2 = u$ .... $((s+)$ and $(r_=+))$,

(8) $\vdash$ **begin** $x_1 := x;\ x_2 := 0$ **end** (**if** $x_2 = 0$ **then** $x_3 := x_1$ **else** $x_3 := k(x_1, x_2 - 1) + 1)$ $(x_3 + 2 = u)$ .... $((r_{cv}+), (B+))$,

(9) $\vdash$ **begin** $(x_3 := x)$ **end** $(x_3 + 2 = u)$ .... $((k+)_{\mathcal{M}})$,

(10) $\vdash x + 2 = u$ .... $((s+))$.

By point 5 (i) of the RETRPROV-algorithm we put $u = x + 2$ into X. By point 7 of this algorithm we get $\mathcal{A}_{\mathcal{M}} = \{s \in Seq' : u = x + 2 \in \text{right}(s)\}$. So (10) becomes a special axiom. We can see that our algorithm does not simply calculate an expression, but instead it tries to prove it. We get the expression $x + 2 = u$ as a solution of the functional equation $k(x, 2) = u$, which certainly is not only a calculation. ■

Example 16.

Let us consider the following expression:

(1) $\vdash \forall_{x_2}((p_1(x_2) \to \exists_x p_2(x)) \equiv \exists_{x_1}(p_1(x_2) \to p_2(x_1)))$

Of course we present only a sketch of the proof of the sequent (1).

(2) $\vdash x_2 := y(((p_1(x_2) \to \neg\forall_x \neg p_2(x)) \to \neg\forall_{x_1} \neg(p_1(x_2) \to p_2(x_1))) \wedge$ $(\neg\forall_{x_1} \neg(p_1(x_2) \to p_2(x_1)) \to (p_1(x_2) \to \neg\forall_x \neg p_2(x))))$ .... $(\forall +)$,

(3) $\vdash x_2 := y(\neg\forall_{x_1} \neg(p_1(x_2) \to p_2(x_1)) \to (p_1(x_2) \to \neg\forall_x \neg p_2(x)))$ .... $(C+)$,

(4) $\vdash x_2 := y((p_1(x_2) \to \neg\forall_x \neg p_2(x)) \to \neg\forall_{x_1} \neg(p_1(x_2) \to p_2(x_1))$ .... $(C+)$,

(3.1) $\vdash x_2 := y(\forall_{x_1} \neg(p_1(x_2) \to p_2(x_1))),\ x_2 := y(p_1(x_2) \to \neg\forall_x \neg p_2(x))$ .... $(I+)$, $(\text{-}N)$,

(3.2) $x_2 := y(p_1(x_2)),\ x_2 := y(\forall_x \neg p_2(x)) \vdash x_2 := y(x_1 := y_1(\neg(p_1(x_2) \to p_2(x_1)))$ .... $(I+)$, $(N+)$, $(\forall +)$,

(3.3) $p_1(y),\ x_2:=y(\forall_x\neg p_2(x)),\ \neg p_2(t_1),\ x_2:=y(x_1:=y_1(p_1(x_2)\to p_2(x_1))\vdash$
.... (-∀), (-s), (N+),

(3.4) $p_1(y),\ x_2:=y(\forall_x\neg p_2(x)),\ p_2(y_1)\vdash p_2(t_1)$ .... (I+), (-N), (-s),

(3.5) $p_1(y),\ x_2:=y(\forall_x\neg p_2(x))\vdash p_2(t_1),\ p_1(y)$ .... (I+), (-N), (s+),
which is an axiom by unication,

(4.1) $x_2:=y(p_1(x_2)\to\neg\forall_x\neg p_2(x)),\ x_2:=y(\forall_{x_1}\neg(p_1(x_2)\to p_2(x_1)))\vdash$
.... (I+), (N+),

(4.2) $x_2:=y(\forall_{x_1}\neg(p_1(x_2)\to p_2(x_1))),\ (y_2:=t_2(x_2:=y(x_1:=y_2(\neg(p_1(x_2)\to$
$p_2(x_1)))))\vdash p_1(y)$ .... (-∀), (-I), (s+),

(4.3) $x_2:=y(\forall_{x_1}\neg(p_1(x_2)\to p_2(x_1))),\ (y_2:=t_2(x_2:=y(x_1:=y_2(\neg(p_1(x_2)\to$
$p_2(x_1)))))\vdash x_2:=y(x:=y_3(\neg p_2(x))$

(4.2.1) $x_2:=y(\forall_{x_1}\neg(p_1(x_2)\to p_2(x_1))),\ p_1(y)\vdash p_2(t_2),\ p_1(y)$
.... (-N), (I+), (s+), (-s), which is an axiom,

(4.3.1) $x_2:=y(\forall_{x_1}\neg(p_1(x_2)\to p_2(x_1))),\ p_1(y),\ p_2(y_3)\vdash p_2(t_2)$
.... (-N), (I+), (N+), (-s), (s+) which is an axiom by unification. ■

Example 17.

Let us consider the following formula:
$f(0)=1\wedge(\forall_n(\neg(n=0)\to(f(n)=n*f(n-1)\to f(n+1)=(n+1)*f(n))))$
where f was defined in Example 11.

In some cases we are able to use the mathematical induction, e.g. in this example we proved the correctness of a program defining the factorial. It means as well that we proved (this proof is too long so we omitted it) the equivalence between two descriptions of factorial: the mathematical definition and the program. The mathematical definition is as follows:

$$n! = \begin{cases} 1 & \text{when } n=0 \\ n*(n-1)! & \text{when } n>0 \end{cases} \qquad (ii)$$

and the program is given in Example 11. This proof shows the correctness of the program. ■

Example 18.

If we want to calculate the value of the expression of the form:
$\frac{l_1}{m_1}+\frac{l_2}{m_2}$ where l_1, l_2, m_1, m_2 denote the variables of the type of integer, first we calculate the value $m3:=NWW(m_1,m_2)$ where $NWW(m_1,m_2)$ is the least, common multiple of m_1 and m_2. Next we calculate the values $l_1:=l_1*\frac{m_3}{m_1}$, $l_2:=l_2*\frac{m_3}{m_2}$ and $l_3:=l_1+l_2$. Obviously the result of the division during the calculation of l_1 and l_2 is integer. Hence $\frac{l_3}{m_3}$ is the result of the considered expression. During this calculation we considered the function $NWW(x,y)$. Now we show how we can use the rules of decomposition especially the rules $(\neg k)_{\mathcal{M}}$ and $(k+)_{\mathcal{M}}$. Let $NWW(x_1,y_1)=K_oz$ where K_o means:

```
begin
    x := x1; y := y1;
    while ¬(x = y) do
    if x > y then
                x := x − y;
```

else

$\qquad y := y - x;$

$w := x;$

$z := x_1 * (y_1/w);$

end

We shall show how to get $NWW(6, 4)$ in the standard model of arithmetic. We have to calculate the value $(K_o z)_R(v_1)$ where v_1 is the valuation such that $v_1(x_1) = 6$ $v_1(y_1) = 4$. This valuation is represented by the substitution $s =$ **begin** $x_1 := 6;\ y_1 := 4$ **end**. We decompose the program **begin** $s;\ K_o$ **end** to the normal assignment

$[sK_o] = [[x_1/6, y_1/4], [[x/x_1, y/y_1], [*[\neg(x = y) \veebar [(x > y)[x/x - y][y/y - x]]], [w/x],$
$[z/(x_1 * (y_1/w))]]]] \,{}^{\wedge 1}\!\!\prec [[[x_1/6, y_1/4], [x/x_1, y/y_1]], [*[\neg(x = y) \veebar [(x > y)[x/x - y][y/y - x]]],$
$[w/x], [z/x_1 * (y_1/w))]]] \,{}^{\wedge 4}\!\!\prec [[[x_1/6, y_1/4], [x/x_1, y/y_1]], (v[\veebar[(x > y)[x/x - y][y/y - x]],$
$*[\neg(x = y) \veebar [(x > y)[x/x - y][y/y - x]]])^{\bullet} [[w/x], [z/(x_1 * (y_1/w))]]] \stackrel{\text{Def.2.4(i)(ii)}}{=} [[[x_1/6, y_1/4],$
$[x/x_1, y/y_1]], [\veebar[(x > y)[x/x - y][y/y - x]], [*[\neg(x = y) \veebar [(x > y)[x/x - y][y/y - x]]],$
$[[w/x], [z/(x_1 * (y_1/w))]]]]] \,{}^{\wedge 3}\!\!\prec [[[x_1/6, y_1/4], [x/x_1, y/y_1]], (v[x/x - y])^{\bullet}[*[\neg(x = y) \veebar [(x > y)$
$[x/x - y][y/y - x]]], [[w/x], [z/(x_1 * (y_1/w))]]]] \stackrel{\text{Def.2.4.(i)(ii)}}{=} [[[x_1/6, y_1/4], [x/x_1, y/y_1]],$
$[[x/x - y], [*[\neg(x = y) \veebar [(x > y)[x/x - y][y/y - x]]], [[w/x], [z/(x_1 * (y_1/w))]]]]] \,{}^{\wedge 1}\!\!\prec$
$[[[x_1/6, y_1/4], [x/x_1, y/y_1]]^{\bullet}[x/x - y], [*[\neg(x = y) \veebar [(x > y)[x/x - y][y/y - x]], [[w/x],$
$[z/(x_1 * (y_1/w))]]]] \stackrel{\text{Def.2.4(ii)}}{=} [[[x_1/6, y_1/4], [[x/x_1, y/y_1], [x/x - y]]] [*[\neg(x = y) \veebar [(x > y)$
$[x/x - y][y/y - x]], [[w/x], [z/(x_1 * (y_1/w))]]]] \,{}^{\wedge 4}\!\!\prec [[[x_1/6, y_1/4], [[x/x_1, y/y_1], [x/x - y]]],$
$[[w/x], [z/x_1 * (y_1/w))]]] \,{}^{\wedge 1}\!\!\prec [[[x_1/6, y_1/4], [[x/x_1, y/y_1], [x/x - y]]]^{\bullet}[w/x], [z/(x_1 * (y_1/w))]]$
$\stackrel{\text{Def.2.4(ii)}}{=} [[[x_1/6, y_1/4], [[x/x_1, y/y_1], [x/x - y]]^{\bullet}[w/x]], [z/(x_1 * (y_1/w))]] \stackrel{\text{Def.2.4(ii)}}{=} [[[x_1/6, y_1/4],$
$[[x/x_1, y/y_1], [[x/x - y], [w/x]]]], [z/(x_1 * (y_1/w))]] \,{}^{\wedge 1}\!\!\prec [[x_1/6, y_1/4], [[x/x_1, y/y_1], [[x/x - y],$
$[w/x]]]]^{\bullet}[z/(x_1 * (y_1/w))] \stackrel{\text{Def.2.4(ii)}}{=} [[x_1/6, y_1/4], [[x/x_1, y/y_1], [[x/x - y], [[w/x]], [z/(x_1 *$
$(y_1/w))]]]]].$

Let us denote the normal assignment $[[x_1/6, y_1/4], [[x/x_1, y/y_1], [[x/x - y], [[w/x], [z/(x_1 * (y_1/w))]]]]]$ by symbol Σ_o. We proved by using R_{trans} that the program $[sK_o]$ can be decomposed to the normal assignment Σ_o. Therefore $[sK_o] \prec \Sigma_o$. By Theorem 10 we get $[sK_o]_R(v) = \Sigma_{oR}(v)$. Hence $(K_o z)_R(v_1) = z_R(K_R(v_1)) = z_R(K_{oR}(s_R(v))) = z_R([sK_o]_R(v)) = z_R(\Sigma_{oR}(v)) = 6 * (4/(6 - 4)) = 12$.

We proved that the function $NWW(6, 4)$ defined by program has the value 12. Therefore RETRPROV-algorithm during the prove $\vdash NWW(6, 4) = u$ finds a special set of axioms $\mathscr{A}_{AR} = \{s \in Seq' : 12 = u \in right(s)\}$. ■

Let us remark that the RETRPROV system described above definitely differs from **RS**-algorithm. The main difference concerns the set of rules (rules of decomposition) and the RETRPROV-algorithm. The key idea lies in the rules $(k+)_{\mathscr{M}}$ and $(-k)_{\mathscr{M}}$, since from Corollary 2 and the model of arithmetic, the expression $(sK)^{\mathscr{M}}$ being the result of the execution of program enables us to optimize the calculation. It is also worth emphasizing that RETRPROV produces results in an evidently shorter and faster way (see Example 7 and Example 11 or Example 10 (iv) and Example 13), the correctness of which is guaranteed by Corollary 2.

Chapter 7

Summary and concluding remarks

7.1 Conclusion

The main result of the first part of this paper is algorithmic structural completeness of algorithmic logic strengthened by the substitution rule i.e. the derivability of all structural, finitary and admissible rules. To date the well-known substitution rules which were considered do not fit into these considerations, since they do not preserve the properties of programs. The substitution rule which was defined in this paper was just enough strongly deductive to allow to prove algorithmic structural completeness of algorithmic logic. This result enables us to use many structural and finitary rules in practice provided admissibility of them is proved. Some examples of such rules:

1. $\dfrac{\mathcal{S}(\textbf{while } \alpha \textbf{ do } K)\ TRUE,\ \mathcal{S}\ M\ TRUE,\ \mathcal{S}\ L\ TRUE}{\mathcal{S}(\textbf{while } \alpha \textbf{ do begin } M;\ K;\ L \textbf{ end } TRUE)}$,
   where $(\vartheta(M) \cup \vartheta(L)) \cap (\vartheta(\alpha) \cup \vartheta(K)) = \emptyset$ and $\mathcal{S}$ denotes any finite sequence of substitutions.
2. $\dfrac{\alpha}{p(\alpha)}$, where α is a classical open formula and p is any program-substitution.
   Let us take $p \in Sb$ wich is defined by $g \in G$, classical open formula λ and by the functions e and e_0 such as in Theorem 9. We get from α the formula $p(\alpha)$ equal $h^e(\alpha)$ which is equivalent $(\overline{s_Y\alpha} \wedge \lambda) \vee (h^{e_0}(\alpha) \wedge \neg\lambda)$ for s_Y being such as in Theorem 9.
3. $\dfrac{\mathcal{S}(\alpha \rightarrow \beta)}{\mathcal{S}((\textbf{while } \alpha \textbf{ do } K)\ TRUE \rightarrow (\textbf{while } \beta \textbf{ do } K)\ TRUE}$,
   where $\mathcal{S}$ denotes any finite sequence of substitutions.

Such rules allow to simplify the proofs of correctness of programs.

It is worth to pay attention to the following questions:

(1°) Is it true that the property of structural completeness of some logic is a result of the completeness of this logic i.e. of the property of the form $C_R(X) = C^{\models}(X)$ for every set of formulas X.

(2°) Is it true that the property of Post-incompleteness of algorithmic logic (i.e. $C_{R^*}(\{\alpha\}) \neq F$ for some $\alpha \notin C_{R^*}(\emptyset)$) strengthened by the substitution rule is a result of the completeness of this logic for the set of rules R.

To answer these questions it is not enough to know whether the property of completeness holds.

Let us consider the point (1°). There exists a consequence which is complete but is not structurally complete. Such a system is for example the classical logic with quantifiers based on the set of axioms A_Q and on the set of rules $R_{o\forall} = \{r_o, r_\forall\}$, where r_o is the modus ponens rule and $r_\forall$ is the generalization rule of the form:

$$r_o : \frac{\alpha, \alpha \to \beta}{\beta}, \qquad r_\forall : \frac{\alpha}{\forall_x \alpha}.$$

Indeed, such a system is not structurally complete. To illustrate that let us consider the rule r defined as follows:

$<\{\alpha\}, \beta> \in r$ iff there exists $e \in \mathscr{E}$ such that:

$\alpha = \neg\forall_x\forall_y(e(P(x)) \to e(P(y)))$ and $\beta = \forall_x \neg(e(P(x)) \to e(P(x)))$,

where $\mathscr{E}$ is the set of substitutions defined by W. A. Pogorzelski and T. Prucnal [71].

The admissible rules in the logic with programs are called permissible in the logic without programs. The definition of the latter is analogous to the former one. The rule r is admissible in the considered logic since α is not valid in any interpretation, in any model with a single element therefore $\alpha \notin C_{R_{o\forall}}(A_Q)$ for every $e \in \mathscr{E}$. Since the antecedent of the definition of permissibility is false therefore the rule r is permissible in this logic. Moreover r is structural. However r is not derivable in this logic since in the opposite case for $e \in \mathscr{E}$ such that $e(P(x)) = P(x)$ and $e(P(y)) = P(y)$ by the deduction theorem, we get $\exists_x\exists_x(P(x) \wedge \neg P(y)) \to \forall_x \neg(P(x) \to P(x)) \in C_{R_{o\forall}}(A_Q)$ which is not true. Therefore the structural completeness is not an immediate result of the property of completeness. □

Now we consider the point (2°). Some variant of the question (2°) was known i.e. the logic without the substitution rule is incomplete. However, with regard to the study of the structural rules and the extension of algorithmic logic to the algorithmically structurally complete logic it appeared that the introduction of the notion of the substitution rule was the sufficient condition. After introducing a new substitution rule with a very strong deduction we get the algorithmic structural completeness of algorithmic logic strengthened by the substitution rule. Therefore, the algorithmic logic strengthened by the substitution rule could become complete. So we have to prove the incompleteness of such extended algorithmic logic. The proof of incompleteness of this logic could be done in a different manner if we knew that the substitution rule was hereditary in every model (i.e. if $<\{\alpha\}, \beta> \in r_*$ and α is valid in a model then $\beta(\beta = p(\alpha))$ is valid in this model too). But we think that it is not easy to prove this property without our considerations. The difficulties depend on changing the shape of formulas by using the substitution $p \in Sb$ which is defined by e fulfilling some properties (see Definition 3, 8 and Theorem 9). For example: $e(\rho(\tau_1, ..., \tau_n)) = \rho(\tau'_1, ..., \tau'_n) \wedge \lambda$.

It is worth to notice that the theorem on algorithmic structural completeness allows us to use many secondary rules in various considerations. The only condition which such a rule ought to fulfil is to be structural, finitary and admissible in algorithmic logic strengthened by the substitution rule. This condition is in a way a useful criterion for using many secondary rules.

There is an interesting and open problem of getting structural completeness without assuming finitary rules. □

The second part of this paper is devoted to the construction of proving algorithms. The first of these algorithms called **RS**-algorithm use Gentzen's method and some idea of decomposition of formulas containing programs. We use some P. Gburzyński's ideas [28], [29] connected with proving theorems without programs but we make it possible i.e. we extend these ideas essentially to a case of algorithmic logic i.e. we can prove algorithmic formulas and test programs and their properties for example the correctness and equivalence of some programs and we can consider functions and relations defined by programs. At the time the existing implementation of proving theorems was not able to achieve that. Therefore our implementation was the first one serving programs. The new created **RS**-algorithm in a sense enables us to execute an expert's report since it answers the question whether some relation $\rho(x, y)$ defined by a generalized formula holds. For example if $\rho(x, y)$ is defined by $K_2 b$ which expresses the order relation between natural numbers $x = 1$ and $y = 2$ then we ask the question by writing for example $\rho(1, 2) \equiv b$ and we understand this expression as follows: for which b the relation $\rho(1, 2)$ holds. The proof depends on assuming b to be $TRUE$ when the relation $\rho(x, y)$ holds or to be FALSE otherwise. For example if $x = 1$ and $y = 2$, the algorithm tries to prove the expression $\rho(1, 2) \equiv b$ by replacing $\rho(1, 2)$ by the program $K_2 b$ defining this relation. Finally using the rules of decomposition we get the sequent $TRUE \models b$ which ends the proof by adding the special axiom $\{s \in Seq' : b \in right(s)\}$ to the set of axioms.

The action of RS-algorithm cannot be treated as a calculation of the program since for the program K_4 of the form: **begin** $i := i + 3$; $z := x$ **end** and for the function $g(x)$ defined by $K_4 z$, the algorithm gives for the equality $g(n)^4 = u$ the axiom i.e. the set of sequents $\{s \in Seq' : u = n^4 \in right(s)\}$ as the solution to this equality.

It is known that without changing variables into values the standard calculation of function g is impossible. It is worth to notice that the considered RS-algorithm gives us the result even in case when the standard calculation overflows the stack. To explain that let us consider the program of the form: **if** $x = 0$ **then** $z := 2$ **else** $z := h(x - 1, h(x, y))$; denoted earlier by K_5 and the function h(x, y) defined by $K_5 z$.

We shall try to calculate the value of the function $h(1, 2)$ during the execution of RS-algorithm and in fact we shall try to prove the equation $h(1, 2) = u_3$. RS-algorithm finds the solution $u_3 = 2$ though the shape of the program defining the function h evidently makes it impossible for compiler since the compilation leads to overflowing.

The mere process of dynamic looking for the set of axioms is more complicated than it seems from this short description. The example of this can be a test of the proof of the expression $f(2) = u$ for $f(2)$ defined by $K_1 z$. During the proof there appeare two sequents. The first of them leads to the sequent of the form: $\models 0 = u, 1 = u, 2 = u$, which at first sight makes the further proof impossible since we have difficulties with choosing the proper assumption among the expressions of the form: $u = 0$, $u = 1$ and $u = 2$. The second sequent leads during further decomposition to the sequent of the form: $0 = 0 \models 2 = 0, 1 = 0, 2 = u$ which becomes an axiom after assuming that each sequent s for which $2 = u \in right(s)$ is an axiom. Lemma 8 enables us to solve this problem. It ensures that for every leaf of the tree which unables us the univocal choice of the specific axiom (the sequent containing the expression of the form $u = \tau$ on the right side of the symbol $\models$) among many such expressions occuring in this leaf there exists another leaf in which this choice is univocal i.e. there exists only one expression of the form $u = \tau$ on the right side of the symbol $\models$. After this choice the earlier leaf in which appears diversity of meaning becomes an axiom too.

RS-algorithm contains also a particular manner to avoid the difficulties which appear during the decomposition of the sequent containing the program with the word **WHILE**. The rule generates in one case the infinite set of sequents and only the special treatment of this case enables us effective activity of **RS**-algorithm.

The expression of the form: W, s **while** α **do** $K\ \beta \models Y, b, Z$ is changed by the rule $(+k)$ into the formula $W, s(p := TRUE) \bigcup$ **begin** $p := p \wedge \alpha$; K **end**$(p \wedge \neg\alpha \wedge \beta) \models Y, b, z$. For further considerations we denote by $M_n(l)$: **begin** s; $p := TRUE$ **end** [**begin** $p := p \wedge \alpha$; K **end**]^l where l means a natural number.

As a result of using the rule $(+\bigcup)$ we ought to consider the set $\{M_n(l)(p \wedge \neg\alpha \wedge \beta), W \models Y, b, Z : l \in \mathcal{N}\}$. Since it is impossible to construct all of these elements in practice therefore we further consider the sequent of the form: $M_n(l)(p \wedge \neg\alpha \wedge \beta), W \models \Gamma$ where Γ denotes the expression: Y, b, Z. This sequent is treated in a special way by **RS**-algorithm in the point 4.

RS-algorithm enables us to prove the correctness of some programs with STOP property. Moreover this algorithm eliminates the inconsistence of the definition of relation. Let us assume the following definition: $\rho(x) \equiv \neg\rho(x)$.

If we want to know whether the relation holds **RS**-algorithm starts the proof of the expression of the form: $\models \rho(x) \equiv b$. Since during the execution of **RS**-algorithm we meet the expressions b and $\rho(x)$ on the same side of the symbol $\models$ therefore **RS**-algorithm STOP and we get an answer about the inconsistence of the above definition of $\rho(x)$ since only the expression with negation is able to change the side of the symbol $\models$.

Besides the Gentzen's method we considered in our paper a sequential method of the decomposition of programs. This method decomposes each program with STOP property in the model $\mathcal{M}$ into a normal assignment which can be executed on terms or on formulas. The assumption of STOP property of the program means in implementation the possibility of execution of all needed calculations in this program. Under such an assumption there is no problem with the decomposition of the considered program.

References

[1] Banachowski L: *Investigations of properties of programs by means of the extended algorithmic logic.* I. Ann. Soc. Math. Pol., Ser. IV. Fundamenta Informaticae, Vol. I. 1, 93—119 (1977).

[2] Bartol W. M., Gburzyński P., Findeisen P., Kreczmar A., Lao M., Litwiniuk A., Müldner T., Nykowski W., Oktaba H., Salwicki A., Szczepańska-Waserstrum D.: *Loglan. International Summer School of the programming language. Zaborów. Poland, September, 5—10.1983.* Institute of Informatics, University of Warsaw. Warszawa 1983.

[3] Bibel W.: *Automated Theorem Proving* 2., rev. ed. Braunschweig—Wiesbaden—Vieweg 1987 (Artificial intelligence).

[4] Biela A.: *Program-substitution and admissibility of rules in algorithmic logic.* Acta Informatica, 25, 439—473 (1988).

[5] Biela A.: *Retrieval system and dynamic algorithm looking for axioms of notions defined by programs.* Fundamenta Informaticae, Vol. 19, No. 3 (1993).

[6] Biela A., Dziobiak W.: *On two properties of structurally complete logics.* Reports on Math. Logic, 16, 51—54 (1982).

[7] Biela A., Borowczyk J.: *RETRPROV: a system that looks for axioms.* Acta Informatica (1995).

[8] Biela A., Wojtylak M.: *Automatyczne dowodzenie twierdzeń.* Wyd. Uniw. Śląskiego, Katowice 1993.

[9] Blake A.: *Canonical expressions in Boolean algebra.* PhD thesis, Univ. of Chicago, Illinois 1937.

[10] Blasius K., Eisinger N., Siekmann J., Smolka G., Herold A., Walter C.: *The Markgraf Karl Refutation Procedure. Proceedings of the IJCAI-81*, 1981, pp. 511—518.

[11] Bledsone W. W., Tyson M.: *The UT interactive Prover*, MEMO ATP-17, Math. Dept. Univ. of Texas, May 1975.

[12] Boyer R. S., Moore J. S.: *A Computational Logic.* ACM Monograph. Academic Press, New York 1979.

[13] Boyer R. S., Moore J. S.: *A verification condition generator for FORTRAN.* Academic Press, London 1981.

[14] Bundy A.: *Catalogue of Artificial Intelligence Tools.* Springer—Verlag 1986.

[15] Cardelli L.: *ML under Unix.* Bell Laboratories, Mursay Hill, New Jersey 1982.

[16] Chang W., Lee R.: *Symbolic Logic and Mechanical Theorem Proving.* Academic Press, New York—San Francisco, London 1973.

[17] Chlebus B.: *On the decidability of propositional algorithmic logic.* Zeitschr. für Math. Logik und Grundlagen der Math., 28, 247—261 (1982).

[18] Church A.: *Introduction on Mathematical Logic.* Princeton 1956.

[19] Cooper D.C.: *Theorem proving in computers. Advanced in programming and non-numerical computation,* ed. Fox, Pergamon Press, Oxford 1966, pp. 155—182.

[20] Dańko W.: *A criterion of undecidability of algorithmic theories.* Ann. Soc. Math. Pol., Ser. IV. Fundamenta Informaticae, 3, 605—628 (1981).

[21] Dańko W.: *Algorithmic properties of finitely generated structures. Proceedings, Poznań, August 1980.* Lect. Notes in Computer Science, Vol. 148: *Logic of programs and their applications.* Springer, Berlin 1983, pp. 118—131.

[22] Dańko W.: *Definability in algorithmic logic.* Ann. Soc. Math. Pol., Ser. IV. Fundamenta Informaticae. II, 277—287 (1979).

[23] Davis M.: *A computer program for Presburger's procedure.* Summaries of talks presented at the Summer Institute for Symbolic Logic (1957). Second edition published by Institute for Defense Analysis, Princeton NJ 1960.

[24] Davydov G. V.: *Synthesis of the resolution method with the inverse method.* J. Soviet Math., 1, 12—18 (1973).

[25] Dummet M.: *Elements of intuitionism.* Clarendon Press, Oxford 1977, p. 169.

[26] Dunham B., North J.: *Theorem testing by computer. Proc. Symp. Mathem. "Theory of Automata".* Polytechnic Press, Brooklyn NY 1963, pp. 173—177.

[27] Engeler E.: *Algorithmic properties of structures.* Math. Syst. Theory, 1, 183—195 (1967).

[28] Gburzyński P.: *Badania eksperymentalne w dziedzinie automatycznego dowodzenia twierdzeń. Analiza porównawcza dwóch metod.* Praca doktorska. IIUW, Warszawa 1982, pp. 1—57.

[29] Gburzyński P.: *Mechanical proving system of universal purpose.* Raport CO-PAN nr 390, 1980.

[30] Gentzen G.: *Untersuchungen über das logische Schliessen.* Math. Zeitschr., 39, 176—210, 405—431 (1935).

[31] Gordon M. J. C.: *Representing a logic in the lcf metalanguage.* In: *Tools and Notions for Program Construction.* Ed. D. Neel. Cambridge University Press, Cambridge 1982.

[32] Gordon M., Milner A., Wadsworth C.: Edinburgh LCF: *A Mechanized Logic of Computation.* Lect. Notes in Computer Science, 78, Springer—Verlag, Berlin 1979.

[33] Green C.: *Theorem proving by resolution as a basis for question — answering systems.* In: *Machine Intelligence 4.* Eds. B. Meltzer, D. Michie. Edinburgh University Press, Edinburgh 1969.

[34] Greenbaum S.: *Input transformations and resolution implementation techniques for theorem proving in first order logic.* Ph.D. thesis. University of Illinois at Urbana Champaign 1986.

[35] Greenbaum S., Plaisted D.: *The Illinois prover: a general purpose resolution theorem prover, 8th Conference on Automated Deduction 1986.*

[36] Grundy J., Newey M.: *Theorem proving in higher order logics. 11th International Conference, TPHOLs'98, Canberra, Australia, September 27—October 1.* Lect. Notes in Computer Science, 1479, 1998, 497 pp.

[37] Guard J., Oglesby F., Bennett J., Settle L.: *Semi-automated mathematics,* JACM, 18, 49—62 (1969).

[38] Haan J., Schubert L. K.: *Inference in a topically organized semantic net. Proceedings of the AAAI-86.* 1986, pp. 334—338.

[39] Harel D., Pratt V.: *Nondeterminism in Logics of programs. Proc. 9-th Ann. ACM Symp. on Theory of Computing, Boulder, Colorado, May 1977.* MIT, Cambridge, MA 1977.

[40] Harrison J.: *Theorem proving with real numbers.* Logical Found. of Computer Sciences & Mathematical Logic, 12, 186 (1998).

[41] Herbrand J. J.: *Recherches sur la theorie de la demonstration.* Travaux Soc. Sci. et Lettres Varsovie, Cl.3 (Math., Phys.), 128 (1930)

[42] Hermes H.: *Introduction to mathematical logic.* Springer, Berlin 1973.

[43] Hilbert D., Ackermann W.: *Grundzüge der theoretischen Logik.* Springer, Berlin 1967.

[44] Hines L. M.: *Building in Knowledge of Axioms,* Ph. D. Dissertation, Univ. of Texas.

[45] Hines L. M.: *Hyper-chaining and Knowledge-based Theorem Proving.* CADE-9. 1986.

[46] Jansohn H. S., Landwehr R., Wrigtson G.: *An Interactive Proof System for Higher Order Logic. Proceedings of the 5th European Meeting on Cybernetics and System Research, 1980.*

[47] Jansohn H. S., Landwehr R., Wrigtson G.: Design. *Implementation and Results of an Interactive Proof System for Higher Order Logic.* Universität Karlsruhe, Interner Bericht 19/79, Karlsruhe 1979.

[48] Kreczmar A.: *Effectivity problems of algorithmic logic.* Ann. Soc. Math. Pol., Ser. IV. Fundamenta Informaticae, Vol. I.1, 19—32 (1977).

[49] Kreczmar A.: *The set of all tautologies of algorithmic logic is hyperarithmetical.* Bull. Acad. Polon. Sci., Ser. Math., 21, 781—783 (1971).

[50] Loveland D. W.: *Automated Theorem Proving: A Logical Basis.* Fundamental Studies in Computer Science, Vol. 6, 406 (1978) (ISBN 0-7204-0499-1, North-Holland).

[51] Loveland D. W.: *A linear format for resolution. Symp. on Automatic Demonstration.* Lect. Notes in Math., 125. Springer, Berlin, pp. 147—162.

[52] Loveland D. W.: *A simplified format for the model elimination procedure.* JACM, 16, 349—363 (1969).

[53] Loveland D. W.: *A unifying view of some linear Herbrand procedures.* JACM, 19, 366—384 (1972).

[54] Luckham D.: *Refinement theorems in resolution theory. Symp. Automatic Demonstration.* Lect. Notes in Mathem., 125. Springer, Berlin 1970, pp. 163—190.
[55] Lusk E. L., McCune W. W., Overbeek R. A.: *Logic machine architecture: kernel functions. D.W. Loveland 6th Conference on Automated Deduction.* Lect. Notes in Computer Science, 138. Springer—Verlag, Berlin 1982.
[56] Miller S. A., Schubert L. K.: *Using Specialists to Accelerate General Reasoning. Proceedings of the AAI-88*, pp. 161—165.
[57] Minc G.: *Proizvodnost dopustimych pravil. Issliedovanija po konstruktivnoj matiematikie i matiematičeskoj logikie.* Vol. 5. Matiematičeskij Institut im. V. A. Steklova. Leningrad 1972, pp. 85—89.
[58] Mirkowska G.: *Algorithmic logic and its applications in the theory of programs.* I. Ann. Soc. Math. Pol., Ser IV. Fundamenta Informaticae, Vol. I.1, 1—17 (1977).
[59] Mirkowska G.: *Algorithmic logic and its applications in the theory of programs* II. Ann. Soc. Math. Pol., Ser. IV. Fundamenta Informaticae. Vol. I.2, 147—165 (1977).
[60] Mirkowska G.: *Algorithmic logic with nondeterministic programs.* Ann. Soc. Math. Pol., Ser. IV. Fundamenta Informaticae 3, 45—64 (1980).
[61] Mirkowska G.: *Model existence theorem in algorithmic logic with non-deterministic programs.* Ann. Soc. Math. Pol., Ser. IV. Fundamenta Informaticae 3, 157—170 (1980).
[62] Mirkowska G.: *On formalized systems of algorithmic logic.* Bull. Acad. Sci., Ser. Math., 19, 421—428 (1971).
[63] Mirkowska G., Orłowska E.: *An elimination quantifiers in a certain class of algorithmic formulas.* Ann. Soc. Math. Pol., Ser. IV. Fundamenta Informaticae I (1978), 347—355.
[64] Mirkowska G., Salwicki A.: *Algorithmic logic.* PWN. D. Reidel Publishing Company, Warszawa 1987.
[65] Newell A., Shaw J., Simon H.: *Empirical explorations of the logic theory machine. Proc. West. Joint Computer Conf.* Vol. 15. 1957, pp. 218—239.
[66] Ohlbach H. J.: *Link Inheritance in Abstract Clause Graphs.* J. Automated Reasoning, 3, 1—34 (1987).
[67] Paulson L.: *Natural deduction as higher-order resolution.* J. Logic Programming, 3, 237—258 (1986).
[68] Perkowska E.: *On algorithimic m-valued logics.* Bull. Acad. Polon. Sci., Ser. Sci. Math. Astr. Phys., 20, 717—719 (1972).
[69] Petermann U.: *On algorithmic Logic with partial operations. Proceedings, Poznań, August 1980.* Lect. Notes in Computer Science, 148. *Logic of programs and their applications.* Springer, Berlin 1983, pp. 213—223.
[70] Pogorzelski W. A.: *Structural completeness of the propositional calculus,* Bull. Acad. Polon. Sci., Ser. Sci. Math. Astr. Phys., 19, 349—351 (1971).
[71] Pogorzelski W. A., Prucnal T.: *Structural completeness of the first order predicate calculus.* Zeitschr. für Math. Logik und Grundlagen der Math., 21, 315—320 (1975).
[72] Porte J.: *Antitheses in systems of relevant implication.* J. Symb. Logic, 48, 97—99 (1983).

[73] Pratt V.: Semantical Considerations on Floyd-Hoare Logic. In: *Proceedings 17-th Ann. IEEE. Symp. on FCS, October 1976*, pp. 109—121.
[74] Prawitz D.: *An improved proof procedure.* Theoria, 26, 102—139 (1960).
[75] Prawitz D., Prawitz H., Voghera N.: *A mechanical proof procedure and its realization in an electronic computer.* JACM7, 102—128 (1960).
[76] Prucnal T.: *Structural completeness of Lewis's system S5.* Bull. de l'Acad. Polon. des Sci., Ser. Sci. Math. Astr. Phys., 29, 101—103 (1972).
[77] Radziszowski S.: *Programmability and $P = NP$ conjecture. Proc. FCT' 77.* Cont. Lect. Notes in Computer Science, 56. Springer, Berlin 1977.
[78] Raph K. M. G.: *The Markgraf Karl refutation procedure, Seki-84-08-kl.* Fachbereich Informatik, Universität Kaiserslautern 1984.
[79] Quine W. V.: *A way to simplify truth functions.* American Math. Monthly 62, 627—631 (1955).
[80] Rasiowa H.: *On logical structure of programs.* Bull. Polon. Acad. Sci., Ser. Math. Astr. Phys., 20, 319—324 (1972).
[81] Rasiowa H., Sikorski R.: *Mathematics of Metamathematics.* PWN, Warsaw 1968.
[82] Rasiowa H.: *ω^+-valued algorithmic logic as a tool to investigate procedures. Proc. MFCS'74.* Lect. Notes in Computer Science. Vol. 28. Berlin, Heidelberg: Springer 1974.
[83] Robinson J.: *A Machine-oriented Logic Based on the Resolution Principle.* JACM 12, 23—41 (1965).
[84] Robinson J.: *Logic: Form and function.* University Press, Edingburgh 1979.
[85] Robinson J. A.: *Mechanizing HOL.* Machine Inteligence 4. Edinburgh Univ. Press. Edinburgh 1969.
[86] Rusinoff D.: *An experiment with the Boyer-Moore theorem prover: A proof of Wilson's theorem.* J. Automated Reasoning, 1, 121—139 (1985).
[87] Salwicki A.: *Axioms of Algorithmic Logic univocally determine Semantics of programs.* Proc. MFCS'80. Lect. Notes in Computer Science 88. Springer, Berlin 1980, pp. 352—361.
[88] Salwicki A.: *Formalized algorithmic languages.* Bull. Acad. Pol. Sci., Ser. Sci. Math. Astr. Phys., 18, 227—232 (1970).
[89] Salwicki A.: *Programmability and recursiveness, an application of algorithmic logic to procedures.* Dissertation. Uniwersytet Warszawski, Warszawa 1976.
[90] Segerberg K.: *A Completeness Theorem in Modal Logic of Programs* (abstract). Notices of the American Mathematical Society, October 1977.
[91] Stickel M. E.: *An introduction to Automated Deduction. Fundamentals of Artificial Intelligence. An Advanced Course.* Lect. Notes in Computer Science, 232. Springer—Verlag, Berlin 1986, pp. 75—132.
[92] Stickel M. E.: *A Prolog Technology Theorem Prover: implementation by an extended Prolog compiler. Eighth International Conference on Automatic Deduction.* Lect. Notes in Computer Science, 230. Springer—Verlag, Berlin 1986, pp. 573—587.
[93] Stickel M. E.: *Automated deduction by theory resolution.* J. Automated Reasoning, 1, 4, 333—355 (1985).
[94] Stickel M. E.: *Automatic Deduction by Theory Resolution. Proceedings of the IJCAI-85.* (1985), pp. 1181—1186.

[95] Szałas A.: *Algorithmic logic with recursive functions*. Ann. Soc. Math. Pol., Ser. IV. Fundamenta Informaticae, 4, 975—995 (1981).
[96] Thiele H.: *Wissenschaftstheoretische Untersuchungen in algoritmischen Sprachen.* VEB Deutscher Verlag der Wissenschaften, Berlin 1996.
[97] Trybulec A.: *Język informacyjno-logiczny MIZAR-MSE.* Prace IPI PAN, ICS PAN REPORT, No. 465, PAN, Warszawa 1982.
[98] Tsitkin A. I.: *On structurally complete superintuitionistic logics.* Dokl. AN SSSR 241, Moskwa 1978, pp. 40—43.
[99] Wang T. C.: *Designing examples for semantically quided hierarchical deduction. Proceedings of the IJCAI-85*, 1985, pp. 1201—1207.
[100] Wang T. C., Bledsone W. W.: *Hierarchical deduction.* J. Automated Reasoning 3, 35—77 (1987).
[101] Wang H.: *Proving theorem by pattern recognition.* I. Comm. of ACM, 3, 220—234 (1960).
[102] Wang H.: *Proving theorem by pattern recognition. II.* Bell System Technical Journ., 40, 1—41 (1961).
[103] Wang H.: *Toward Mechanical Mathematics.* IBM Journ. of Research and Development, 4, 2—22 (1960).
[104] Wojtylak P.: *On structural completeness of many valued logic's.* Studia Logica, 3, 3—8 (1974).

Andrzej Biela

Algorytmiczna strukturalna zupełność i system wyszukiwania dowodów twierdzeń w teoriach algorytmicznych

Streszczenie

Dowody poprawności oprogramowania są jedynym sposobem zapewnienia użytkownika (inwestora), że można z niego korzystać bez ryzyka. W pracy rozważa się zatem klasę reguł algorytmicznie strukturalnie zupełnych, pozwalających na poprawne wnioskowanie.

Duże znaczenie w automatycznym dowodzeniu twierdzeń ma właściwy dobór reguł, dlatego badania rozpoczęto od próby uzasadnienia wyprowadzalności reguł dopuszczalnych w logice algorytmicznej.

W publikacji zawarto wyniki badań dotyczące algorytmicznej strukturalnej zupełności logiki algorytmicznej oraz omówiono system automatycznego dowodzenia twierdzeń, w którym pewne relacje czy funkcje mogą być reprezentowane za pomocą programów. Badania przedstawiono w języku umożliwiającym wyrażenie własności programów (rozdz. 2).

Pierwsza część pracy dotyczy:

1) wprowadzenia reguły podstawiania do logiki algorytmicznej i do logiki z niedeterministycznymi programami oraz udowodnienia zasadniczych własności podstawiania (rozdz. 3),

2) uzasadnienia algorytmicznej strukturalnej zupełności logiki algorytmicznej z dołączoną regułą podstawiania (rozdz. 4).

Zdefiniowano zbiór podstawień taki, że wprowadzona za jego pomocą reguła podstawiania okazała się, mówiąc intuicyjnie, na tyle „silna dedukcyjnie", iż pozwoliła na uzyskanie algorytmicznej strukturalnej zupełności logiki algorytmicznej. Na podstawie tej własności stwierdza się, że w konsekwencji logiki algorytmicznej każda reguła strukturalna, finitarna i dopuszczalna jest w niej wyprowadzalna. Można zatem swobodnie stosować reguły z tej klasy. Ponadto dla niezupełnego systemu logiki algorytmicznej otrzymano pewien rodzaj *quasi*-zupełności, którym jest algorytmiczna strukturalna zupełność.

Dalszą część pracy (rozdz. 5) poświęcono omówieniu systemu dowodzącego, który umożliwia dowodzenie twierdzeń metodą Gentzena, sformułowanych w języku różnych teorii, a także dowodzenie twierdzeń o programach. Ponadto możliwe są dowody wyrażeń nie będących twierdzeniami, polegające na znalezieniu i dołączeniu dodatkowych aksjomatów umożliwiających dowód. System ten pozwala również na dowodzenie poprawności programów, rozwiązywanie równań funkcyjnych, których funkcje są zdefiniowane za pomocą programów, a także badanie relacji zdefiniowanych za pomocą procedur oraz badanie niezależności aksjomatów.

Pragnąc potwierdzić wiarygodność teoretycznych rozważań, system ten został zaimplementowany w języku LOGLAN, a następnie w języku PASCAL i z jego wykorzystaniem wykonano liczne eksperymenty. Niektóre z nich zostały zaprezentowane w podrozdz. 5.6.

Była też możliwa inna metoda dowodu wyzyskująca model arytmetyki, dlatego rozdz. 6 zawiera opis tej metody, polegający na rozkładzie programów. W książce podano aksjomaty

rozkładu i twierdzenie gwarantujące sprowadzenie każdego programu z własnością STOP-u w rozważanym modelu do podstawienia będącego wynikiem tego rozkładu. Reguły omawianego systemu dowodzącego posługują się wynikiem będącym podstawieniem, a nie samym programem, co znacznie upraszcza dowód. W rozdziale 7 omówiono główne idee przedstawione w pracy.

Andrzej Biela

Алгоритмическая структурная полнота и система находки и доказательства теорем в алгоритмических теориях

Резюме

В автоматичном доказательстве теорем большое значение имеет соответствующий отбор правил, поэтому эти исследования начали с испытания доказательства выведения правил допустимых в алгоритмической логике. В работе содержатся итоги исследований, касающиеся алгоритмической структурной полноты алгоритмической логики, а также обсуждена в ней система автоматичного доказательства теорем, в которой некоторые отношения или функции могут быть представлены с помощью программ. Эти исследования представлены на языке алгоритмической логики, делающем возможным, выражение собственности программ (глава 2).

Первая часть работы касается:

1) введения правила подстановки в алгоритмическую логику и логику с недетерминистическими программами а также доказательства принципиальных свойств подстановки (глава 3).

2) доказательства алгоритмической структурной полноты алгоритмической логики с приложенным правилом подстановки (глава 4).

Тем самым определили такое множество подстановок, что позволило получить структурную полноту алгоритмической логики. Эта собственность устанавливает, что в результате алгоритмической логики каждое структурное, финитарное и допустимое правило является выведенным. Таким образом можно свободно применять правила из этого класса. Кроме того для неполной системы алгоритмической логики получен некоторый род *quasi*-полноты, которым является алгоритмическая структурная полнота.

Следующая часть работы (глава 5) посвящена обсуждению доказывающей системы, которая делает возможным доказательство теорем методом Гентцена сформулированных на языке разных теорий, тоже доказательство теорем содержащих программы. Сверх того возможны доказательства выражений не являющихся теоремами, заключающиеся в находке и приложении добавочных аксиом делающих возможным доказательство. Эта система делает возможным тоже доказательство правильности программ, решение функциональных уравнений, которых функции определены с помощью программ, тоже исследование определённых связей с помощью процедур а также исследование самостоятельности аксиом.

Целью представления теоретических рассуждений эта система заимплементована на языке ЛОГЛАН а затем ПАСКАЛ и с его помощью сделали ряд экспериментов. Некоторые из них были представлены в главе 5.6.

Так как возможна была другая метода доказательств при использовании модели арифметики, поэтому в главе 6 содержится описание этого метода, заключающегося

в расписании программ. Представлены там аксиомы расписания и теорема гарантирующая приведение каждой программы со свойством STOP-а в рассуждённой модели к подстановке являющейся результатом этого рассписания. Тем самым правила рассуждаемой доказывающей системы пользуются результатом, являющимся подстановкой, а не самой программой, что значительно сокращает доказательство.