STUDIES IN LOGIC, GRAMMAR AND RHETORIC 8 (21) 2005

Anna Zalewska
University of Białystok

INTRODUCTORY REMARKS ON INFERENCE RULES FOR ALGORITHMIC LOGIC WITH PROCEDURES

1. Algorithmic logic (AL) ([2]) supplies a set of logical axioms and inference rules appropriate for reasoning about properties of programs. The problem of AL axiomatization was completely solved for programs without procedures. A certain proof system for AL with program variables has been presented in [3]. It bases on a suitable Gentzen-type axiomatization and uses the notion of a tree of sequents as a basis tool. In this paper we consider a certain extension of the proof system in which algorithmic properties of programs with simple procedures (without procedure parameters) can be proved. We add new constructs to our algorithmic language (blocks and procedures) and appropriate new Gentzen-like rules of inference. According to [1] we treat procedure text as a text constant and declaration of procedure as an assignment of this text constant to the name of the procedure.

2. Procedures allow us to express in evident way a program structure by logical closed elements. Procedures are named sequences of instructions. The connection between the name and the sequence of instructions is expressed by the following procedure declaration:

procedure name;
declaration of local variables
begin
procedure_body
end {name}.

The declaration of procedure (without parameters) consists of two parts: the procedure header (i.e. the keyword **procedure** and the procedure identifier: name) and the procedure text (i.e. instructions between keywords **begin** and **end**). In order to indicate that the procedure instructions

ISBN 83-7431-059-6 ISSN 0860-150X

should be done in a given point of the program it is enough to write procedure identifier. Some examples of procedure declarations are presented below.

EXAMPLES.

The example of simple procedure declaration without local variables.

```
procedure exchange;
begin
 t:=x;
 x:=y;
 y:=t;
end {exchange}
```

The example of simple procedure declaration with a local variable.

```
procedure exchange;
 var t:integer;
begin
 t:=x;
 x:=y;
 y:=t;
end {exchange}
```

The example of procedure declaration with nested procedure declaration.

```
procedure exchange;
 var t:integer;
 procedure add;
  var t:integer;
 begin
  t:=x;
  x:=x+y;
  y:=x+t;
 end {add};
begin
 t:=x;
 x:=y;
 y:=t;
 add;
end {exchange}.
```

3. The algorithmic logic is an extension of the first-order logic by the expressions of the following form

$$M\alpha$$

where M is:

- *a program variable*;
- *assignment statement*: (x:=τ) or (q:=γ) where x is an individual variable, τ is a classical term, γ is an open formula and q is a propositional variable;
- *composed program*: **begin** M; M' **end**;
- *branching program*: **it** γ **then** M **else** M' **fi**;
- *iteration program*: **while** γ **do** M **od** where γ is an open formula and M and M' are programs;
- *block*: **beginblock** D; I_1; ... I_n **endblock** where D is a declaration of local variables and procedures without parameters and $I_1; \ldots I_n$ are instructions.

Some examples of blocks are presented below.

EXAMPLES.

The example of simple block with a procedure declaration without local variables.

beginblock
procedure exchange; **begin** t:=x; x:=y; y:=t; **end** {exchange};
exchange;
endblock;

The example of block with a procedure declaration (with a local variable).

beginblock
procedure exchange; **var** t:integer;
begin t:=x; x:=y; y:=t; **end** {exchange}
exchange;
endblock;

The example of block with procedure declaration (with nested procedure declaration).

beginblock
procedure exchange; **var** t:integer;
procedure add; **var** t:integer; **begin** t:=x; x:=x+y; y:=x+t; **end** {add};
begin t:=x; x:=y; y:=t; add; **end** {exchange};
exchange;
endblock;

In general we are basing on the notion of realization of language and valuation of variables given in [1, 3]. Programs are interpreted as partial functions. The informal meaning of the formula M is "*after execution of the program M the formula α holds*".

4. We are basing on the Gentzen-like system given in [3]. In the system each of the decomposition rules describes relation between its conclusion (written over a line) and its premise or premises (written under the line):

$$\frac{\text{conclusion}}{\text{premise}_1;\ \text{premise}_2;\ \ldots\ \text{premise}_n}$$

The set of inference rules is extended by the following schemes;

I) The scheme of the inference rule for simple block with a procedure declaration without local variables.

$$\frac{\begin{array}{l}\{\Gamma, s\ \textbf{beginblock}\\ \quad \textbf{procedure}\ \text{name\_p}_1;\ \textbf{begin}\ \text{body\_p}_1\ \textbf{end}\ \{\text{name\_p}_1\};\\ \quad \ldots\ldots\ldots\\ \quad \textbf{procedure}\ \text{name\_p}_n;\ \textbf{begin}\ \text{body\_p}_n\ \textbf{end}\ \{\text{name\_p}_n\};\\ \quad I_1;\ \text{name\_p}_1;\ \ldots\ I_n;\ \text{name\_p}_n;\ I_{n+1};\\ \textbf{endblock}\ \alpha, \Delta\}\end{array}}{\{\Gamma, s\ \textbf{begin}\ I_1;\ \text{body\_p}_1;\ \ldots\ I_n;\ \text{body\_p}_n;\ I_{n+1};\ \textbf{end};\ \alpha, \Delta\}}$$

II) The scheme of the inference rule for block with a procedure declaration (with a local variable).

$$\frac{\begin{array}{l}\{\Gamma, s\ \textbf{beginblock}\\ \quad \textit{declaration of local variables}\ x_1, \ldots, x_m;\\ \quad \textbf{procedure}\ \text{name\_p}_1;\ \textbf{begin}\ \text{body\_p}_1\ \textbf{end}\ \{\text{name\_p}_1\};\\ \quad \ldots\ldots\ldots\\ \quad \textbf{procedure}\ \text{name\_p}_n;\ \textbf{begin}\ \text{body\_p}_n\ \textbf{end}\ \{\text{name\_p}_n\};\\ \quad I_1;\ \text{name\_p}_1;\ \ldots\ I_n;\ \text{name\_p}_n;\ I_{n+1};\\ \textbf{endblock}\ \alpha, \Delta\}\end{array}}{\{\Gamma, s\ \textbf{begin}\ I'_1;\ \text{body\_p}'_1;\ \ldots\ I'_n;\ \text{body\_p}'_n;\ I_{n+1};\ \textbf{end}\ \alpha, \Delta\}}$$

where x'_j $(j = 1, \ldots, m)$ does not occur on the lefthand side of the sequent
$I'_i = I_i(x_j/x'_j)$
$\text{body\_p}'_i = \text{body\_p}_i(x_j/x'_j)$

III) The scheme of the inference rule for block with procedure declaration (with nested procedure declaration).

$\{\Gamma, s$ **beginblock**
procedure name_p;
declaration of local variables $x_1, \ldots, x_m$;
procedure name_np;
declaration of local variables $y_1, \ldots, y_k$;
begin body_np **end** {name_np};
begin IP_1; body_np; IP_2; **end** {name_p};
I_1; name_p; I_2; **endblock** $\alpha, \Delta\}$

$\{\Gamma, s$ **begin** I_1'; IP_1'; body_np'; IP_2'; I_2' **end** $\alpha, \Delta\}$

where x_j' $(j = 1, \ldots, m)$ does not occur on the lefthand side of the sequent
y_j' $(j = 1, , k)$ does not occur on the lefthand side of the sequent
body_np' = body_np$(x_j/x_j', y_j/y_j')$
body_p' = body_p(x_j/x_j')
$IP_i' = I_i(x_j/x_j', \ , y_j/y_j')$ $(i = 1, 2)$
$I_i' = I_i(x_j/x_j')$ $(i = 1, 2)$

5. Some examples of applications of the above rules are given below.

I)

$\{(t = 0) \wedge (x = 1) \wedge (y = 2) \rightarrow$
beginblock
procedure exchange; **begin** t:=x; x:=y; y:=t; **end** {exchange};
exchange; **endblock** $((t = 1) \wedge (x = 2) \wedge (y = 1))\}$

$\{(t \neq 0), (x \neq 1), (y \neq 2),$
beginblock
procedure exchange; **begin** t:=x; x:=y; y:=t; **end** {exchange};
exchange; **endblock** $((t = 1) \wedge (x = 2) \wedge (y = 1))\}$

$\{(t \neq 0), (x \neq 1), (y \neq 2),$
begin t:=x; x:=y; y:=t **end** $((t = 1) \wedge (x = 2) \wedge (y = 1))\}$

$\{(t \neq 0), (x \neq 1), (y \neq 2),$ (t:=x)(x:=y)(y:=t) $((t = 1) \wedge (x = 2) \wedge (y = 1))\}$

$\{(t \neq 0), (x \neq 1), (y \neq 2),$ (t:=x)(x:=y) $((t = 1) \wedge (x = 2) \wedge (t = 1))\}$

$\{(t \neq 0), (x \neq 1), (y \neq 2),$ (t:=x) $((t = 1) \wedge (y = 2) \wedge (t = 1))\}$

$\{(t \neq 0), (x \neq 1), (y \neq 2), (x = 1) \wedge (y = 2) \wedge (x = 1)\}$

$\{(t \neq 0), (x \neq 1), (y \neq 2), (x = 1)\} \mid \{(t \neq 0), (x \neq 1), (y \neq 2), (y = 2)\}$
$\mid \{(t \neq 0), (x \neq 1), (y \neq 2), (x = 1)\}$

II)

$\{(t = 0) \wedge (x = 1) \wedge (y = 2) \rightarrow$
beginblock
var t: integer;
procedure exchange; **begin** t:=x; x:=y; y:=t; **end** {exchange};
exchange; **endblock** $((t = 0) \wedge (x = 2) \wedge (y = 1))\}$

$\{(t \neq 0), (x \neq 1), (y \neq 2),$
beginblock
var t: integer;
procedure exchange; **begin** t:=x; x:=y; y:=t; **end** {exchange};
exchange; **endblock** $((t = 0) \wedge (x = 2) \wedge (y = 1))\}$

$\{(t \neq 0), (x \neq 1), (y \neq 2),$
begin t':=x; x:=y; y:=t' **end** $((t = 0) \wedge (x = 2) \wedge (y = 1))\}$

$\{(t \neq 0), (x \neq 1), (y \neq 2),$ (t':=x)(x:=y)(y:=t') $((t = 0) \wedge (x = 2) \wedge (y = 1))\}$

$\{(t \neq 0), (x \neq 1), (y \neq 2),$ (t':=x)(x:=y) $((t = 0) \wedge (x = 2) \wedge (t' = 1))\}$

$\{(t \neq 0), (x \neq 1), (y \neq 2),$ ((t':=x) $((t = 0) \wedge (y = 2) \wedge (t' = 1))\}$

$\{(t \neq 0), (x \neq 1), (y \neq 2),$ $((t = 0) \wedge (y = 2) \wedge (x = 1)\}$

$\{(t \neq 0), (x \neq 1), (y \neq 2), (t = 0)\} \mid \{(t \neq 0), (x \neq 1), (y \neq 2), (y = 2)\}$
$\mid \{(t \neq 0), (x \neq 1), (y \neq 2), (x = 1)\}$

III)

$\{(t = 0) \wedge (x = 1) \wedge (y = 2) \rightarrow$
beginblock
procedure exchange; **var** t:integer;
procedure add; **var** t:integer;
begin t:=x; x:=x+y; y:=x+t; **end** {add};
begin t:=x; x:=y; y:=t; add; **end** {exchange};
exchange;
endblock; $((t = 0) \wedge (x = 3) \wedge (y = 5))\}$

$\{(t \neq 0), (x \neq 1), (y \neq 2),$
beginblock
procedure exchange; **var** t:integer;
procedure add; **var** t:integer;
begin t:=x; x:=x+y; y:=x+t; **end** {add};
begin t:=x; x:=y; y:=t; add; **end** {exchange};
exchange;
endblock; $((t = 0) \wedge (x = 3) \wedge (y = 5))\}$

$\{(t \neq 0), (x \neq 1), (y \neq 2),$
begin t':=x; x:=y; y:=t'; t":=x; x:=x+y; y :=x+t"
end $((t = 0) \wedge (x = 3) \wedge (y = 5))\}$

$\{(t \neq 0), (x \neq 1), (y \neq 2),$
(t':=x)(x:=y)(y:=t')(t":=x)(x:=x+y)(y:=x+t")
$((t = 0) \wedge (x = 3) \wedge (y = 5))\}$

$\{(t \neq 0), (x \neq 1), (y \neq 2),$
(t':=x)(x:=y)(y:=t')(t":=x)(x:=x+y) $((t = 0) \wedge (x = 3) \wedge (x + t'' = 5))\}$

$\{(t \neq 0), (x \neq 1), (y \neq 2),$
(t':=x)(x:=y)(y:=t')(t":=x) $((t = 0) \wedge (x + y = 3) \wedge (x + y + t'' = 5))\}$

$\{(t \neq 0), (x \neq 1), (y \neq 2),$
(t':=x)(x:=y)(y:=t') $((t = 0) \wedge (x + y = 3) \wedge (x + y + x = 5))\}$

$\{(t \neq 0), (x \neq 1), (y \neq 2),$ (t':=x)(x:=y) $((t = 0) \wedge (x + t' = 3) \wedge (x + t' + x = 5))\}$

$\{(t \neq 0), (x \neq 1), (y \neq 2),$ (t':=x) $((t = 0) \wedge (y + t' = 3) \wedge (y + t' + y = 5))\}$

$\{(t \neq 0), (x \neq 1), (y \neq 2),$ $(t = 0) \wedge (y + x = 3) \wedge (y + x + y = 5)\}$

$\{(t \neq 0), (x \neq 1), (y \neq 2), (t = 0)\} \mid \{(t \neq 0), (x \neq 1), (y \neq 2), (y + x = 3)\}$
$\mid \{(t \neq 0), (x \neq 1), (y \neq 2), y + x + y = 5)\}$

6. In the paper we give some introductory remarks on inference rules for algorithmic logic with simple procedures. The extended in this way Gentzen-like system is sound and for propositional part of algorithmic logic is complete. The system ([3]) for the first-order algorithmic logic can be extended by inference rules for procedures with parameters called by value or variable. The declaration of such procedures c be expressed in one of the following ways:

declaration of procedures with parameters called	
by variables	**by value**
procedure name(**var** x, **var** y); *declaration of local variables* **begin** *procedure_body* **end** {name}.	**procedure** name(x,y); *declaration of local variables* **begin** *procedure_body* **end** {name}.

References

[1] Mirkowska G., *A Complete Axiomatic Characterization of Algorithmic Properties of Block-Structured Programs with Procedures*, in: it Proc. MFCS'76 (A. Mazurkiewicz ed.), LNCS 45, Springer Verlag, 602–606

[2] Mirkowska G., Salwicki A., *Algorithmic Logic*, D. Reidel Publishing Company, Dordrecht, 1987

[3] Zalewska A., *Program Verification with Algorithmic Logic*, Philomath, Warsaw 2001

Anna Zalewska
University of Białystok
Department of Computer Science
e-mail: zalewska@uwb.edu.pl