On Correct Procedure Parameter Transmission in Higher Programming Languages

by

Hans Langmaack

Institut für Angewandte Mathematik
und Informatik der Universität des
Saarlandes
6600 Saarbrücken 15, Im Stadtwald
Bundesrepublik Deutschland

August 1972
A 72/3

This paper is part of an invited lecture given at the first annual congress of the Gesellschaft für Informatik (GI) in October 1971 in Munich. The paper has been submitted for publication in "Acta Informatica".

S u m m a r y

The paper starts with the observation that in ALGOL 60 no specifications for formal procedure parameters are prescribed, whereas ALGOL 68 demands complete specifications. As a consequence, every ALGOL 68 program accepted by the compiler does not have wrong parameter transmissions at run time whereas ALGOL 60 programs may have them. The property of ALGOL 60 programs to have only correct parameter transmissions obviously is undecidable if all data, conditional statements, etc. have to be taken into consideration and it is unfair to demand that the compiler shall figure out these programs by a finite process. Therefore, we investigate this question of decidability under a much fairer condition, namely to take into consideration no data and no conditions and to give all procedure calls occurring in the same block equal rights. Even this fairer problem turns out to be algorithmically unsolvable, in general (Theorem 3), but it is solvable as soon as the programs do not have global formal procedure parameters (Theorem 1). Analogous answers can be given to the problems of formal equivalence of programs and of formal reachability, formal recursivity, and strong recursivity of procedures (Theorems 5-8). Procedures which are not strongly recursive have great importance in compilation techniques as is shown in Section X.

I. Introduction

This paper deals with the question whether formal parameters of procedures in high level programming languages should be specified or not. The situation is well known: In ALGOL 60 no specification is prescribed, whereas ALGOL 68 demands specifications for all formal parameters, even specifications for the formal parameters of formal procedures etc. must be given by the programmer. PL/1 takes a position in between: Formal parameters of non-formal procedures must be specified, but formal parameters of formal procedures cannot be specified. This means practically that PL/1 in this respect is closer to ALGOL 60 than to ALGOL 68. For, when translating a call of a non-formal or formal ALGOL 68 procedure the compiler is exactly informed about the specifications for all formal parameters. Best possible code can be implemented because superfluous actual data types need not be taken into consideration. As no wrong parameter transmission can happen at run time no run time parameter checks need be implemented. Now, when translating a call of a formal ALGOL 60 or PL/1 procedure the compiler does not know any specification for the formal parameters, so that even actual data types must be taken into account which at run time never occur. Because correct parameter transmission is not completely checked at compile time, run time parameter checks must be provided for.

This short discussion shows that, concerning parameter transmission, ALGOL 68 has clear advantages over the other languages mentioned. On the other hand, concerning parameter transmission, the definition of ALGOL 60 and PL/1 can well be justified if there is an algorithm which for any program at compile time firstly decides whether at run time wrong parameter transmissions might occur and which secondly detects the specifications for all formal procedure parameters. In the following we shall investigate the question in what a sense and under which circumstances such an algorithm exists.

II. Language Limitations

In this paper we will discuss four higher level programming languages:

1) ALGOL 60 without specifications for formal parameters, called ALGOL 60-P (pure),
2) ALGOL 60 with specifications prescribed for formal parameters and denoted in that way indicated in the ALGOL 60 report, called ALGOL 60-PL/1, as PL/1 is covered here,
3) ALGOL 60 with additional specifications for formal parameters for formal procedures, called ALGOL 60-SF, and
4) ALGOL 68.

For our purposes it is useful to have a common frame for all these languages. We choose ALGOL 60 and trim the languages in such a way that they appear as successive restrictions of ALGOL 60-P. For us different languages differ only by the manner how to indicate declarations and specifications (modes). In ALGOL 68 the formal parameters of formal procedures of formal procedures etc. have to be specified. Here, in general, declarations and specifications are trees of declarators, trees which might even be infinite [5,7]. Clearly, these infinite trees must be described in a finite manner. We better call this language ALGOL 60-68.

As an example we present one and the same program Π^1 written in four different languages.

ALGOL 60-P:

```
begin int A;
      proc P(X,Q);
           begin X:=X+1;
                 if X<5 then Q(X,P)
           end;
      A:=1;
      P(A,P);
      print (A)
end
```

ALGOL 60-PL/1:

```
begin int A;
      proc P(X,Q); int X; proc Q;
           begin X:=X+1;
                :
                etc. as above
                :
```

ALGOL 60-SF:

```
begin int A;
      proc P(X,Q); int X;
                   proc (int, proc)Q;
           begin X:=X+1;
                 :
                 etc. as above
                 :
```

ALGOL 60-68:

```
begin int A;
      mode p = proc (int,p);
      proc P(X,Q); int X; p Q;
           begin X:=X+1;
                  :
                  etc. as above
                  :
```

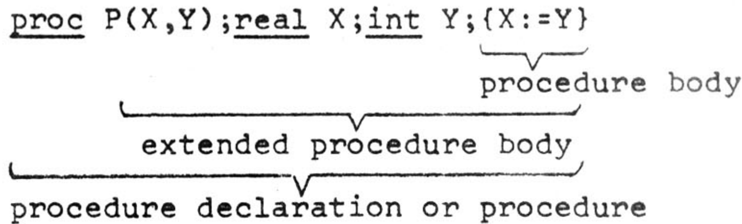
For the aims of this paper it is not necessary to give more precise definitions of the languages. It suffices to be acquainted with ALGOL 60; for our convenience we impose a few restrictions and modifications on ALGOL 60:

a) Only proper procedures, no function procedures are allowed;
b) value listing of formal parameters (in the sense of ALGOL 60) is prohibited;
c) only identifiers are allowed as actual parameters of procedure statements;
d) beside <u>begin</u> and <u>end</u> we have an additional pair of <u>statement braces</u> { }. They act as block-<u>begin</u> and block-<u>end</u> and we require that all procedure bodies are included in these braces. In this context the new statement braces are called <u>body braces</u>.

III. Syntactical and Formal Programs

Definition 1: A syntactical program Π is a string of basic symbols which can be reduced to the axiom <program> by the formal rules of the grammar.

Certain substrings of Π are called blocks, procedure declarations (simply procedures), and procedure bodies. We consider blocks not only to be proper blocks but also the whole program Π, procedure bodies, and so called extended procedure bodies, extended by the formal parameter and specification part, while the declarator proc and the procedure identifier are excluded. Example:



A syntactical program Π can also be considered to be a string

$$\Pi = Z_1 \ Z_2 \ \dots \ Z_n$$

where the symbols Z_i are delimiters, constants, and identifiers. We have to distinguish between an identifier and its occurrence in a program. If Z_i is an identifier then (i,Z_i) is an occurrence of the identifier Z_i in the program $\Pi=Z_1 \ Z_2 \ \dots \ Z_n$. Occurrences of identifiers are declaring or applied. It is well known how to establish in Π a relation δ between an occurrence (i,Z_i) of the identifier Z_i and a declaring occurrence (j,Z_j) with $Z_i=Z_j$.

Definition 2: A syntactical program Π is called formal, if the relation δ is a function, totally defined on the set of all occurrences of identifiers in Π.

If (i,Z_i) is an occurrence of the identifier Z_i then $\delta(i,Z_i)=(j,Z_j)$ is called the associated declaring occurrence of this

identifier. (i,Z_i) is called a bound occurrence, bound by (j,Z_j). If we restrict δ to a block β in Π then δ is still a function, but not necessarily totally defined. If (i,Z_i) is an occurrence in β and $\delta(i,Z_i)$ is undefined in β, i.e. $\delta(i,Z_i)$ occurs outside β, then (i,Z_i) is called a free occurrence of the identifier Z_i in β.

Identifiers in a formal program Π may be renamed. Then $\Pi = Z_1Z_2\ldots Z_n$ becomes $\tilde{\Pi}=\tilde{Z}_1\tilde{Z}_2\ldots\tilde{Z}_n$. A renaming is called admissible if $\tilde{\Pi}$ is a formal program and if for all occurrences (i,Z_i) of identifiers in Π $pr_1(\delta(i,Z_i))=pr_1(\delta(i,\tilde{Z}_i))$ holds. Two formal programs are called identical if they differ only by an admissible renaming of identifiers. A formal program is called distinguished if different declaring occurrences of identifiers $(i,Z_i)\neq(j,Z_j)$ are denoted by different identifiers $Z_i \neq Z_j$. It is clear that in every class of identical formal programs there exists at least one distinguished program.

IV. Compilable Programs

Definition 3: A formal program is called to be correct with respect to compilation or simply compilable if any applied occurrence (i,Z_i) of an identifier, bound by the declaring occurrence $\delta(i,Z_i)=(j,Z_j)$, is applied appropriately according to the declaration.

We do not give a precise definition of the notion appropriate application. For our purpose it is sufficient to know that this relation is a decidable relation between an applied occurrence of an identifier and the associated declaring occurrence. We indicate this by some examples:

a) For operations the modes and the types of the operands must be appropriate. If an operand is formal and no specification is known as in ALGOL 60-P then mode and type of the operand are considered to be appropriate anyway.

b) For assignments the modes and the types of the left and right hand expressions must be appropriate. In the case of a formal left or right hand side the same remark as above in a) holds.

c) For subscripted variables the number of indices must be the same as in the associated array declaration or specification. If the subscripted variable is formal and if we deal with the languages ALGOL 60-P and ALGOL 60-PL/1, nothing is required concerning the number of indices.

d) For procedure statements (calls) the number of parameters must be the same as in the associated procedure declaration or specification. In the case of a formal procedure statement the analogous remark as above in c) holds.

e) If the identifier Z_i of the applied occurrence (i,Z_i) is formal and if there is an associated specification in a specification part, then the application must be appropriate to the specification. If there is no associated specification in a specification part as in ALGOL 60-P then the application is considered to be appropriate anyway as nothing is prescribed by any specification.

f) The mode and the type of an actual parameter of a procedure statement must be appropriate to the specification of the corresponding formal parameter if there is any specification. If there is none as in ALGOL 60-P, then mode and type of the actual parameter are considered to be appropriate anway.

Our example program Π^1 should clarify the meaning of f):
ALGOL 60-P: Actual parameters always fulfill the conditions in f).
ALGOL 60-PL/1: The declarations of the actual parameters X, P,A,P are <u>int</u>, <u>proc</u>(<u>int</u>,<u>proc</u>), <u>int</u>, <u>proc</u>(<u>int</u>,<u>proc</u>), the specifications of the corresponding formal parameters are appropriate, namely empty, empty, <u>int</u>, <u>proc</u>.
ALGOL 60-SF: The declarations of the actual parameters X,P, A,P are <u>int</u>, <u>proc</u>(<u>int</u>,<u>proc</u>(<u>int</u>,<u>proc</u>)), <u>int</u>, <u>proc</u>(<u>int</u>,<u>proc</u> (<u>int</u>,<u>proc</u>)), the specifications of the corresponding formal parameters are appropriate, namely <u>int</u>, <u>proc</u>, <u>int</u>, <u>proc</u>(<u>int</u>, <u>proc</u>).

ALGOL 60-68: The declarations of the actual parameters X,P, A,P are <u>int</u>, <u>p</u>, <u>int</u>, <u>p</u>, the specifications of the corresponding formal parameters are appropriate, they even are identical, i.e. <u>int</u>, <u>p</u>, <u>int</u>, <u>p</u>.

In order to simplify our further discussions with respect to f), we state the following: In ALGOL 60-68 appropriateness means that declarator trees must be identical, in other languages appropriateness means that declarator trees must not be contradictory. If we conceive a declarator tree as a set T of named strings, called nodes, closed under initial segment relation, then two trees are called <u>contradictory</u> if there is a node $n \in T_1 \cap T_2$ with different names: $name_1(n) \neq name_2(n)$.

As long as we deal with the languages ALGOL 60-P, ALGOL 60-PL/1 or ALGOL 60-SF it should be clear that the relation of appropriate application is decidable as all modes are finite. Good compilers for these languages execute this decision properly. For ALGOL 60-68 the decision is not so simple, but can be done, too. Algorithms which effectively detect the identity of modes have been given in [5 , 8 , 10]. As a result, compilability of formal programs is a decidable property for all languages mentioned.

The following example Π^2 may clarify the notion of compilability:

```
begin proc D(x,y); px; qy; { };
      proc M(x,y); px; qy; {x(y)};
      proc M1(x,y);px; qy; {x(D)};
      proc E(η); qη; {η(E,D,D,M1)};
      proc Ē(ξ,α,β,γ); pξ; rα,β,γ; {γ(ξ,Ē)};
      M(E,Ē) end
```

where <u>p</u> stands for <u>proc</u>(<u>proc</u>), <u>q</u> for <u>proc</u>(<u>proc</u>,<u>proc</u>,<u>proc</u>,<u>proc</u>), <u>r</u> for <u>proc</u>(<u>proc</u>,<u>proc</u>).

This program Π^2 is written in ALGOL 60-SF where formal parameters of formal procedures have to be specified. The program is compilable and, consequently, also compilable in ALGOL 60-P resp. ALGOL 60-PL/1 if we drop all specifications for formal

parameters resp. parts of them. But the formal parameters cannot be specified in such a way that the program becomes compilable in ALGOL 68. Otherwise, the following equations would hold:

$$\partial E = \partial x^{M} \;, \quad \partial \bar{E} = \partial y^{M}$$

$$\succ \; \underline{\text{proc}}\;(\partial \eta^{E}) = \underline{\text{proc}}\;(\partial y^{M}) = \underline{\text{proc}}\;(\underline{\text{proc}}(\partial \xi^{\bar{E}}, \partial \alpha^{\bar{E}}, \partial \beta^{\bar{E}}, \partial \gamma^{\bar{E}}))$$

$$\succ \; \partial \gamma^{\bar{E}} \equiv \partial M1 \;, \; \partial \xi^{\bar{\bar{E}}} \approx \partial E$$

$$\succ \; \underline{\text{proc}}\;(\partial \xi^{\bar{E}}, \partial \bar{E}) = \underline{\text{proc}}\;(\partial x^{M1}, \partial y^{M1})$$

$$\succ \; \partial E = \underline{\text{proc}}\;(\partial D)$$

$$\succ \; \partial \eta^{E} = \partial \underline{D}$$

$$\succ \; \underline{\text{proc}}\;(\partial \xi^{\bar{\bar{E}}}, \partial \alpha^{\bar{\bar{E}}}, \partial \beta^{\bar{E}}, \partial \gamma^{\bar{E}}) = \underline{\text{proc}}\;(\partial x^{D}, \partial y^{D})$$

The last equation is a contradiction.

We should not suppress the following remark concerning our definition of correctness with respect to compilation. The definition is based on a sort of local definition of appropriate application of identifier occurrences. If we would demand that the compiler should additionally trace all parameter transmissions, then the program π^{2} could easily be detected to be functionally incorrect. On the other hand it is a crucial question whether we fairly can demand this tracing by a compiler. A compiler is an algorithm which has to give an answer "compilable" or "not compilable" for any submitted formal ALGOL program in a finite time. The tracing of parameter transmissions where all possible input data and all conditional statements must be taken into consideration is an algorithmically unsolvable problem. For it is undecidable whether a given statement in a given formal and even functionally correct program without procedures is superfluous or not.

At best, we can hope to get a positive answer to the following problem: Does tracing of parameter transmissions become an algorithmically solvable task if we allow the compiler to disregard all data and conditional statements? I.e. as soon as a procedure has been called and its body is entered all procedure statements within the main part of the body have

equal rights and are to be processed parallelly. In [4 , 3] there are algorithms which work under this assumption, but they are only sufficient, i.e. their answers are correct only if the answers read "the program has correct parameter transmission".

V. Programs with Correct Parameter Transmissions

Definition 4: A formal program Π is called to be partially compilable if after replacement of all procedure bodies by empty ones the resulting program Π_e is compilable in the sense of Definition 3.

Definition 5: Let Π be partially compilable. A program Π' is called to result from Π by application of the copy rule ($\Pi \vdash \Pi'$) if the following holds:

Let $f(a_1,\ldots,a_m)$ be a procedure statement in the main program of Π. Let

$$\underline{\text{proc}}\ f\ (x_1,\ldots,x_n);\ \underbrace{\sigma}_{\text{specification part}};\ \underbrace{\rho}_{\text{procedure body}};$$

be the associated procedure declaration φ. Partial compilability of Π guarantees that the numbers n of actual and formal parameters are equal. We assume that Π is distinguished. Then $f(a_1,\ldots,a_n)$ is replaced by a modified body ρ', a so called generated block, where the formal parameters x_i occuring in ρ are replaced by the corresponding actual parameters a_i.

Starting from Π' we can easily construct an identical and distinguished program Π'' if we rename all bound occurrences of identifiers in ρ' which are bound within ρ' (local to ρ') by identifiers which do not yet occur in Π'.

$$\begin{array}{l} \Pi\ :\ \ldots;\ \underline{\text{proc}}\ f(x_1,\ldots,x_n);\ \sigma;\ \rho;\ \ldots;\ f(a_1,\ldots,a_n);\ \ldots \\ \vdash \\ \Pi':\ \ldots;\ \underline{\text{proc}}\ f(x_1,\ldots,x_n);\ \sigma;\ \rho;\ \ldots;\ \rho';\ \ldots \end{array}$$

The body braces {} in $\rho = \{\bar{\rho}\}$ become so called call braces in $\rho' = \{\rho'\}$. Let $\overset{+}{\vdash}$ and $\overset{*}{\vdash}$ be the transitive and transitive-reflexive closure of $\vdash$.

Lemma 1: If $\Pi \overset{*}{\vdash} \Pi'$ then Π' is a formal program. $\overset{+}{\vdash}$ resp. $\overset{*}{\vdash}$ are irreflexive resp. reflexive partial orderings in the set of formal programs.

Π' is not necessarily partially compilable even if Π is generally compilable. The programs Π^3

```
begin int a;
      proc p(x,q); {q(x)};
      p(a,p) end
```

resp.

```
begin int a;
      proc p(x,q);int x; proc q; {q(x)};
      p(a,p) end
```

fulfill the conditions of compilability in ALGOL 60-P resp. ALGOL 60-PL/1. But the following programs $\Pi^{3'}$ with $\Pi^3 \vdash \Pi^{3'}$ are not compilable, not even partially:

```
begin int a;
      proc p(x,q); {q(x)};
      {p(a)} end
```

resp.

```
begin int a;
      proc p(x,q); int x; proc q; {q(x)};
      {p(a)} end
```

Specifications in the manner of ALGOL 60-SF do not help either. The program Π^4

```
begin    ⎧ int a;
         ⎪ proc p(q); proc(proc)q; {q(r)};
       Δ ⎨ proc f(x); proc(int) x; {x(a)};
         ⎩ proc r(y); bool y; {y := true};
           p(f) end
```

is compilable, but program $\Pi^{4'}$

```
begin  Δ ; {f(r)} end
```

with $\Pi^{4} \vdash \Pi^{4'}$ is not partially compilable. On the other hand we can state the following:

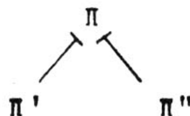
Lemma 2: If Π is a compilable ALGOL 60-68 program and if $\Pi \overset{*}{\vdash} \Pi''$, then Π'' is compilable, too.

Proof: Let $\Pi \vdash \Pi'$. By Definition 5 all formal parameters x_i occuring in ρ are replaced by the corresponding actual parameters a_i. As Π is compilable in ALGOL 68, x_i and a_i have identical declarator trees. Therefore, as the application of x_i is appropriate in Π, the application of a_i in Π' must be appropriate, too. Q.e.d.

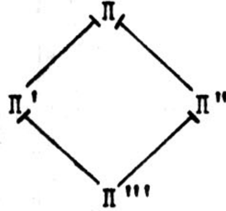
In other languages this conclusion fails as in spite of compilability of Π, x_i and a_i do not necessarily have identical declarator trees. In our example Π^4 q has the declarator tree proc(proc), f has proc(proc(int)). These trees are not contradictory, and they are not identical either. But in $\Pi^{4'}$ the declarator trees of r proc(bool) and of x proc(int)are contradictory.

Lemma 3: If $\Pi \vdash \Pi'$, then the procedure call $f(a_1,...,a_n)$ in Π, mentioned in Definition 5, is uniquely determined by Π,Π'.

Lemma 4: If a diagram



holds, then Π' and Π'' are identical or we can find a program Π''' with



if Π' and Π" are partially compilable. An analogous lemma holds if we invert the arrows.

Lemma 5: If

$$\Pi=\Pi_o \longmapsto \Pi_1 \longmapsto \cdots \longmapsto \Pi_m=\Pi' \text{ and}$$
$$\Pi=\tilde{\Pi}_o \longmapsto \tilde{\Pi}_1 \longmapsto \cdots \longmapsto \tilde{\Pi}_{\tilde{m}}=\Pi'$$

then $m=\tilde{m}$ and one sequence generates the other by successive replacements of partial sequences $\Pi_i \longmapsto \Pi_{i+1} \longmapsto \Pi_{i+2}$ by $\Pi_i \longmapsto \tilde{\Pi}_{i+1} \longmapsto \Pi_{i+2}$.

Definition 6: Let Π be a formal program where { and } are used only as body braces. We call Π an original program. Then

$$E_\Pi := \{\Pi' \mid \Pi \overset{*}{\longmapsto} \Pi'\}$$

is called the execution of Π. Programs Π' in E_Π, different from Π, are called generated programs.

For a generated program Π' in E_Π the following is true: In the main program of Π' outside of any innermost call brace pair all applications of identifiers are appropriate. A program $\Pi' \in E_\Pi$ is maximal if and only if a) Π'_e is compilable and does not contain any procedure statement or b) there is exactly one innermost call brace pair in Π'_e with an inappropriate application of an identifier or c) Π'_e is equal Π_e with an inappropriate application of an identifier. Maximal programs Π' in E_Π of category b) or c) are exactly those, which are not partially compilable.

E_Π contains a tree T_Π the nodes of which are exactly those programs in E_Π with at most one innermost call brace pair. We call T_Π the execution tree of Π. There is a bijection I_Π

from E_Π onto the set $\mathfrak{T}_\Pi$ of all finite subtrees of T_Π, which have at most one program which is not partially compilable:

$$I_\Pi | E_\Pi \xrightarrow[\text{onto}]{1-1} \mathfrak{T}_\Pi .$$

I_Π has the property

$$\Pi' \overset{*}{\longmapsto} \Pi'' \Leftrightarrow I_\Pi(\Pi') \text{ is a subtree of } I_\Pi(\Pi'').$$

Definition 7: An original program Π is called to ~~be~~ formally correct with respect to parameter transmissions or simply to have correct parameter transmissions if all programs in E_Π (or T_Π) are partially compilable.

Lemma 6: If Π has correct parameter transmissions then E_Π is a distributive lattice isomorphic to the lattice of all finite subtrees of T_Π.

Concerning programs Π which have correct parameter transmissions we may give the following remark: Compilable programs $\tilde{\Pi}$ without procedures can be understood to be denotations for transformations $F_{\tilde{\Pi}}$ of states (eventually storage states)

$$F_{\tilde{\Pi}} | \mathfrak{T} \to \mathfrak{T}$$

where $\mathfrak{T}$ is the set of all states. In general, $F_{\tilde{\Pi}}$ is only partially defined. Now, let Π' be a program in E_Π. We may alter Π' into $\tilde{\Pi}'$ by eliminating all procedure declarations and by replacing all remaining procedure statements by

M : goto M

which denotes the totally undefined transformation

$$F^{\emptyset} = \emptyset \subseteq \mathfrak{T} \times \mathfrak{T} .$$

Now, if $\Pi' \longmapsto \Pi''$ then $F_{\Pi'} \subseteq F_{\Pi''} \subseteq \mathfrak{T} \times \mathfrak{T}$. Because E_Π is a lattice, the union

$$F^{*}_{\Pi} := \bigcup_{\Pi' \in E_\Pi} F_{\tilde{\Pi}'}$$

is a partially defined transformation. So a program Π which has correct parameter transmission can be understood to be a denotation for the transformation F^*_Π defined above[1].

VI. Programs without Global Formal Parameters

If $\Pi \vdash\!\!\!-\!\!- \Pi'$, then for all declarations ∂ in Π we have identical copies ∂' in Π'. For all declarations ∂ in the body ρ we have additionally modified copies ∂'_ρ in ρ' as parts of Π'. If $\Pi \vdash\!\!\!\overset{*}{-}\!\!- \Pi'$, i.e. $\Pi = \Pi_0 \vdash\!\!\!-\!\!- \Pi_1 \vdash\!\!\!-\!\!- \cdots \vdash\!\!\!-\!\!- \Pi_n = \Pi'$, $n \geq 0$, then it is clear now, how to define when a declaration ∂' in Π' is called a copy of a declaration ∂ in Π. Let Π be an original program and Π', Π'' programs in E_Π or T_Π. Declarations ∂' in Π'' and ∂'' in Π'' are called similar if they are copies of the same declaration ∂ in Π. A simple inductive argument shows

Lemma 7: Let d be a non-formal identifier occuring within the procedure body of a procedure declaration φ of an original program Π and let d have a declaration ∂. If φ' is a copy of φ in Π', then d has been replaced by the identifier d' of a copy ∂' of ∂ within φ'.

Two nodes Π' and Π'' in T_Π are called similar if their innermost generated blocks (they are endlosed in call braces {}) ρ' and ρ'' differ by renaming of identifiers and if renamed identifiers have similar declarations. The number of similarity classes for nodes in T_Π is limited by

$$M = P \cdot G^F + 1$$

where P is the number of non-formal procedure declarations, G is the number of declaring occurrences of non-formal identifiers, and F is the number of declaring occurrences of formal parameters in Π.

For a given original program Π we can effectively construct the smallest subtree U_Π of T_Π such that every maximal node in U_Π is maximal in T_Π or has a different similar predecessor in U_Π. Paths in U_Π have a length of at most M+1 nodes.

Let (i,x_i) be an applied occurrence of a formal parameter in program Π. If (i,x_i) occurs in the body of ρ of a procedure φ and if $\delta(i,x_i)$ occurs outside φ then x_i is called a <u>global formal parameter</u> of φ. Example:

<u>proc</u> p(x); {<u>proc</u> q(y); {....x....y....}; ....}

x is a global formal parameter of q.

<u>Theorem 1:</u> If an original program Π has no global formal procedure parameters then Π has correct parameter transmission if and only if all programs in U_Π are partially compilable.

<u>Corollary:</u> For original programs Π without global formal procedure parameters it is decidable whether Π has correct parameter transmission or not.

<u>Proof</u> of Theorem 1: Let Π' be a program in T_Π and not in U_Π. Then there is a maximal node Π' in U_Π with

$$\Pi'=\Pi_0 \vdash\!\!\!- \Pi_1 \vdash\!\!\!- \cdots \vdash\!\!\!- \Pi_n = \Pi'', \; n>0, \Pi_\nu \in T_\Pi .$$

We show that all Π_ν are partially compilable and for every Π_ν there is a different similar node $\tilde{\Pi}_\nu$ in U_Π. This assertion is at least true for Π_0. Let it be true for $\Pi_{\nu-1}, 0\leq\nu-1<n$. Then there is a different similar node $\tilde{\Pi}_{\nu-1}$ in U_Π. If $\tilde{\Pi}_{\nu-1}$ is maximal in U_Π it cannot be maximal in T_Π; otherwise, because of the partial compilability, the innermost call brace pair of $\tilde{\Pi}_{\nu-1}$ could not contain any procedure statement which would contradict $\Pi_{\nu-1} \vdash\!\!\!- \Pi_\nu$. So in any case there is a different non-maximal node $\tilde{\tilde{\Pi}}_{\nu-1}$ in U_Π similar to $\Pi_{\nu-1}$. Both, $\Pi_{\nu-1}$ and $\tilde{\tilde{\Pi}}_{\nu-1}$, are partially compilable. Let

$f(a_1,\ldots,a_n)$

be the procedure statement within the main part of the innermost call brace pair of $\Pi_{\nu-1}$ which generates Π_ν. In $\tilde{\tilde{\Pi}}_{\nu-1}$ there is a corresponding procedure statement

$$\tilde{\tilde{f}}(\tilde{\tilde{a}}_1,\ldots,\tilde{\tilde{a}}_n)$$

where $f,\tilde{\tilde{f}},a_1,\tilde{\tilde{a}}_1,\ldots a_n,\tilde{\tilde{a}}_n$ have similar declarations. $\tilde{\tilde{f}}(\tilde{\tilde{a}}_1,\ldots,\tilde{\tilde{a}}_n)$ generates $\tilde{\tilde{\Pi}}'$ in U_Π:

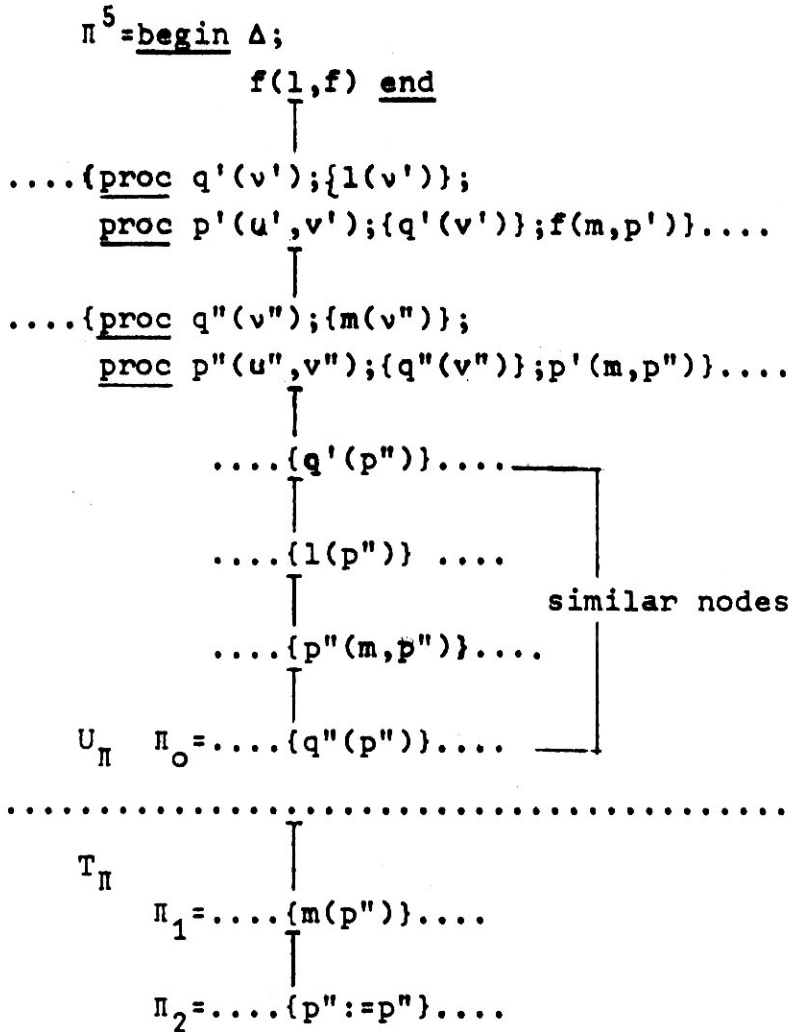
$$\tilde{\tilde{\Pi}}_{\nu-1} \longmapsto \tilde{\tilde{\Pi}}'.$$

The declarations φ and $\tilde{\tilde{\varphi}}$ of f and $\tilde{\tilde{f}}$ are (eventually modified) copies of one and the same declaration $\bar{\varphi}$ of $\bar{f}$ in Π. We have to check by which identifiers the global parameters d, occuring in $\bar{\varphi}$, have been replaced in φ and $\tilde{\tilde{\varphi}}$. d is non-formal by assumption. So, by Lemma 7 in both cases d has been replaced by identifiers having similar declarations. As a consequence the nodes Π_ν and $\tilde{\tilde{\Pi}}'$ are similar and Π_ν is partially compilable, too. So we have proven that Π has correct parameter transmissions. Q.e.d.

The following example Π^5 of an ALGOL 60-P program with global formal parameters shows that the assumption in Theorem 1 is essential:

```
begin ┌ proc l(μ);{μ(m,μ)};
      │ proc m(φ);{φ:=φ};
     Δ< proc f(x,y);
      │     {proc q(ν);{x(ν)};
      └      proc p(u,v);{q(v)};y(m,p)};
        f(l,f) end
```

x is global formal parameter of procedure q. Π^5 has the following trees U_Π and T_Π



All programs in U_Π are partially compilable, nevertheless, Π has incorrect parameter transmission as in Π_2 p" is not applied appropriately. The argumentation in the proof of Theorem 1 fails here because the global parameter x of q has been replaced by l in q' and m in q", where l and m do not have similar declarations.

VII. Equivalent Programs

In order to solve our decision problem for correct parameter transmission we now might ask the following question: Is there an algorithm which transforms any program into an equivalent program without global formal parameters? Equivalence must be defined in such a way that it is invariant with respect to correct parameter transmission.

Let Π be an original program. Let E_Π resp. T_Π be the execution resp. execution tree of Π. For any program $\Pi' \in E_\Pi$ we form the associated main program Π'_m and we replace every remaining procedure statement in Π'_m by a special symbol, say call. This is the reduced main program Π'_r of Π'.

Definition 7: $E_{r\Pi} := \{\Pi'_r \mid \Pi' \in E_\Pi\}$ is the reduced execution of Π. $T_{r\Pi} := \{\Pi'_r \mid \Pi' \in T_\Pi\}$ is the reduced execution tree of Π.

$T_{r\Pi}$ consists of exactly those reduced programs which contain at most one innermost call brace pair. The existence of non-partially compilable programs in T_Π can be recognized in $T_{r\Pi}$ alone:

Π' in T_Π is not partially compilable $\Leftrightarrow$

1) Π'_r is maximal in $T_{r\Pi}$, and
2) the innermost call brace pair of Π'_r has an inappropriate application of an identifier or contains a call-symbol or $\Pi'_r = \Pi_r$ has an inappropriate application of an identifier or contains a call-symbol.

If we define now

Definition 8: Two original programs are called formally equivalent or simply equivalent if their reduced execution trees are identical.

we can prove

Theorem 2: Let the original programs Π_1 and Π_2 be equivalent. Then Π_1 has correct parameter transmissions if and only if Π_2 has correct parameter transmissions, too.

In Definition 8 of program equivalence the term "reduced execution tree" could be replaced by "reduced execution" as the reader may prove.

If two equivalent programs Π_1 and Π_2 have correct parameter transmissions, then they define the same state transformation $F^*_{\Pi_1}=F^*_{\Pi_2}$.

VIII. Undecidabilities

As all our experiments concerning the construction of general algorithms which eliminate global procedure parameters failed, we were led to the conjecture that our decision problem on correct procedure parameter transmission might be unsolvable, in general. So, we will now attack this conjecture by the help of Post's correspondence system[5]. Such a system has two alphabets

$$\mathfrak{A} = \{A,B\},\ A \neq B$$
$$\bar{\mathfrak{A}} = \{\bar{A},\bar{B}\},\ \bar{A} \neq \bar{B},\ \mathfrak{A} \cap \bar{\mathfrak{A}} = \emptyset$$

So we have an isomorphism

$$-\!\!|\ \mathfrak{A} \xrightarrow[\text{onto}]{1-1} \bar{\mathfrak{A}}$$

which we continue to

$$-\!\!|\ \mathfrak{A}^* \xrightarrow[\text{onto}]{1-1} \bar{\mathfrak{A}}^* = \overline{\mathfrak{A}^*}.$$

We consider a production system

$$\Gamma = \{\gamma_1 = (c_1, c_1) = (C_{11} \dots C_{1n_1}, \tilde{C}_{11} \dots \tilde{C}_{1\tilde{n}_1}),$$
$$\vdots$$
$$\gamma_m = (c_m, c_m) = (C_m \dots C_{mn_m}, \tilde{C}_{m_1} \dots \tilde{C}_{m\tilde{n}_m})\}$$

with
$$C_{ij} \in \mathfrak{A},\ \tilde{C}_{ij} \in \bar{\mathfrak{A}}$$
$$\varepsilon \neq c_i \in \mathfrak{A}^*,\ \varepsilon \neq \tilde{c}_i \in \bar{\mathfrak{A}}^*$$
$$n_i \geq 1,\ \tilde{n}_i \geq 1.$$

<u>Definition 9:</u> $\mathcal{L}=(\mathfrak{A},\bar{\mathfrak{A}},\Gamma)$ is called a <u>correspondence system</u> of Post.

We consider non-empty sequences of indices $j_1,\dots,j_r$ with $r\geq 1$, $1\leq j_i\leq m$.

<u>Definition 10:</u> A non-empty sequence of indices is a <u>solution</u> of $\mathcal{L}$:$\Leftrightarrow$ $\overline{c_{j_1}\dots c_{j_r}} = \tilde{c}_{j_1}\dots\tilde{c}_{j_r}$.

<u>Post's Theorem:</u> The property "$\mathcal{L}$ has a solution" is undecidable; in other words: Post's corespondence problem is unsolvable[11].

For any given correspondence system $\mathcal{L}$ of Post we will now effectively construct a compilable ALGOL 60-P program $\Pi_{\mathcal{L}}$ which fulfills the following

<u>Lemma 8:</u> $\mathcal{L}$ has a solution $\Leftrightarrow$ $\Pi_{\mathcal{L}}$ has not correct parameter transmission.

As a consequence we have

<u>Theorem 3:</u> It is undecidable whether a compilable ALGOL 60-P program Π has correct parameter transmission.

For given $\mathcal{L}$ we construct program $\Pi_{\mathcal{L}}$.

```
begin
comment The first part of Π_L is identical for all L ;
proc D(x,y);{};
proc M(x,y);{x(y)};
proc M1(x,y);{ x(D)};
proc E(η);{η(E,D,D,M1)};
proc Ē(ξ;α,β,γ);{γ(ξ,Ē)};
comment The second part of Π_L is different for
        different L . For every j,1≤j≤m we have a
```

procedure L_j. Within L_j we have procedures $C_{j1}[1],\dots,C_{jn_j}[n_j]$ corresponding to the letters C_{ji} in c_j in the production $\gamma_j=(c_j,\tilde{c}_j)$. We have additional procedures $\tilde{C}_{j1}[1],\dots,\tilde{C}_{j\tilde{n}_j}[\tilde{n}_j]$

corresponding to τ_j. As the letters C_{ji} and $C_{ji'}$, $i \neq i'$ might be the same (A or B) we have to distinguish them by indices $[i] \neq [i']$;

$\vdots$

$\underline{\text{proc}}\ L_j\ (x,y);$

$\{\underline{\text{proc}}\ C_{j1}[1](\eta);\{\eta(x,\varkappa_1\langle C_{j1}\rangle,\varkappa_2\langle C_{j1}\rangle,D)\};$

$\underline{\text{proc}}\ C_{j2}[2](\eta);\{\eta(C_{j1}[1],\varkappa_1\langle C_{j2}\rangle,\varkappa_2\langle C_{j2}\rangle,D)\};$

$\vdots$

$\underline{\text{proc}}\ C_{jn_j-1}[n_j-1](\eta);\{\eta(C_{jn_j-2}[n_j-2],\varkappa_1\langle C_{jn_j-1}\rangle,\varkappa_2\langle C_{jn_j-1}\rangle,D)\};$

$\underline{\text{proc}}\ C_{jn_j}\ [n_j](\eta);\{\eta(C_{jn_j-1}[n_j-1],\varkappa_1\langle C_{jn_j}\rangle;\varkappa_2\langle C_{jn_j}\rangle,D)\};$

$\underline{\text{proc}}\ \tilde{C}_{j1}[1](\xi,\alpha,\beta,\gamma);\{\varkappa_3\langle\tilde{C}_{j1}\rangle(\xi,y)\};$

$\underline{\text{proc}}\ \tilde{C}_{j2}[2](\xi,\alpha,\beta,\gamma);\{\varkappa_3\langle\tilde{C}_{j2}\rangle(\xi,\tilde{C}_{j1}[1])\};$

$\vdots$

$\underline{\text{proc}}\ \tilde{C}_{j\tilde{n}_j-1}[\tilde{n}_j-1](\xi,\alpha,\beta,\gamma);\{\varkappa_3\langle\tilde{C}_{j\tilde{n}_j-1}\rangle(\xi,\ \tilde{C}_{j\tilde{n}_j-2}[\tilde{n}_j-2])\};$

$\underline{\text{proc}}\ \tilde{C}_{j\tilde{n}_j}[n_j](\xi,\alpha,\beta,\gamma);\{\varkappa_3\langle\tilde{C}_{j\tilde{n}_j}\rangle(\xi,\ \tilde{C}_{j\tilde{n}_j-1}[\tilde{n}_j-1])\};$

$L_1(C_{jn_j}[n_j],\tilde{C}_{j\tilde{n}_j}[\tilde{n}_j]);\ldots;L_m(C_{jn_j}[n_j],\tilde{C}_{jn_j}[\tilde{n}_j]);M(C_{jn_j}[n_j],\tilde{C}_{j\tilde{n}_j}[\tilde{n}_j])\};$

$\underline{\text{comment}}$ $\varkappa_1\langle C_{ji}\rangle,\varkappa_2\langle C_{ji}\rangle$ are denotations for M, D if $C_{ji}=A$ and for D, M if $C_{ji}=B$. $\varkappa_3\langle\tilde{C}_{ji}\rangle$ is a denotation for α if $\tilde{C}_{ji}=\bar{A}$ and for β if $\tilde{C}_{ji}=\bar{B}$;

$\vdots$

$L_1(E,\bar{E});\ldots;L_m(E,\bar{E})\ \underline{\text{end}}$

<u>Proof</u> of Lemma 8:
Any path in $T_{\Pi_{\mathfrak{L}}}$ starts with a non-empty sequence of calls of L_j:

$$\Pi_{\mathfrak{L}} \vdash^{L_{j_1}} - \Pi_{\mathfrak{L}_{j_1}} \vdash^{L_{j_2}} - \Pi_{\mathfrak{L}_{j_1 j_2}} \cdots \vdash^{L_{j_r}} - \Pi_{\mathfrak{L}_{j_1 \dots j_r}}$$

We describe the structure of the node $\Pi_{\mathfrak{L}_{j_1 \dots j_r}}$ with $r \geq 1$, $1 \leq j_\rho \leq m$.
Let us denote the corresponding strings

$$c_{j_1} \dots c_{j_r} \in \mathfrak{A}^* \text{ and } \tilde{c}_{j_1} \dots \tilde{c}_{j_r} \in \bar{\mathfrak{A}}^*$$

with $\gamma_{j_\rho} = (c_{j_\rho}, \tilde{c}_{j_\rho}) \in \Gamma$ by

$$D_1 \dots D_N \text{ and } \tilde{D}_1 \dots \tilde{D}_{\tilde{N}}$$

with $D_i \in \mathfrak{A}$, $\tilde{D}_i \in \bar{\mathfrak{A}}$, $N = n_{j_1} + \dots + n_{j_r}$

and $\tilde{N} = \tilde{n}_{j_1} + \dots + \tilde{n}_{j_r}$.

The necessary renamings of identifiers in $\Pi_{\mathfrak{L}_{j_1 \dots j_r}}$ can be performed by raising the discriminating indices [i] of

$$C_{ji}[i] \text{ and } \tilde{C}_{ji}[i] \quad .$$

Compared with $\Pi_{\mathfrak{L}}$ $\Pi_{\mathfrak{L}_{j_1 \dots j_r}}$ has the following additional procedure declarations (*)

<u>proc</u> $D_1[1](\eta); \{\eta(E, \varkappa_1 < D_1 >, \varkappa_2 < D_1 >, D)\}$
<u>proc</u> $D_K[K](\eta); \{\eta(D_{K-1}[K-1], \varkappa_1 < D_K >, \varkappa_2 < D_K >, D)\}$
for $K = 2, \dots, N$
<u>proc</u> $\tilde{D}_1[1](\xi, \alpha, \beta, \gamma); \{\varkappa_3 < \tilde{D}_1 > (\xi, E)\}$
<u>proc</u> $\tilde{D}_K[K](\xi, \alpha, \beta, \gamma); \{\varkappa_3 < \tilde{D}_K > (\xi, \tilde{D}_{K-1}[K-1])\}$
for $K = 2, \dots, \tilde{N}$

The main part of the innermost call brace pair of $\Pi_{\mathcal{L}_{j_1 \ldots j_r}}$ has the following procedure statements (**)

$\{..;L_1(D_N[N],\tilde{D}_{\tilde{N}}[\tilde{N}]);\ldots;L_m(D_N[N],\tilde{D}_{\tilde{N}}[\tilde{N}]);M(D_N[N],\tilde{D}_{\tilde{N}}[\tilde{N}])\}$

The proof for (*) and (**) can be given by a simple inductive argument. $\Pi_{\mathcal{L}_{j_1 \ldots j_r}}$ is compilable. Therefore, the only chance to hit a maximal node on a path in $T_{\Pi_{\mathcal{L}}}$ is to call M for a first time. Repetitive calls of L_j lead to an infinite path in $T_{\Pi_{\mathcal{L}}}$.

In a first case we assume that there is a greatest number h with

$$\bar{D}_N=\tilde{D}_{\tilde{N}},\ldots,\bar{D}_{N-h}=\tilde{D}_{\tilde{N}-h}$$
$$h\geq 0,\ N-h\geq 2,\ \tilde{N}-h\geq 2 \quad .$$

Under this assumption we have a path

$$\Pi_{\mathcal{L}_{j_1 \ldots j_r}} = \ldots\{\ldots;M(D_N[N],\tilde{D}_{\tilde{N}}[\tilde{N}])\}\ldots$$
$$\ldots\{D_N[N](\tilde{D}_{\tilde{N}}[\tilde{N}])\}\ldots$$
$$\ldots\{\tilde{D}_{\tilde{N}}[\tilde{N}](D_{N-1}[N-1],\varkappa_1\langle D_N\rangle,\varkappa_2\langle D_N\rangle,D)\}\ldots$$
$$\ldots\{M(D_{N-1}[N-1],\tilde{D}_{\tilde{N}-1}[\tilde{N}-1])\}\ldots$$

which obviously can be prolonged to

$$\ldots\{M(D_{N-h-1}[N-h-1],\tilde{D}_{\tilde{N}-h-1}[\tilde{N}-h-1])\}\ldots$$

We put $\breve{N}=N-h-1$ and $\hat{N}=\tilde{N}-h-1$ if h exists. In the second case, where h does not exist, we put $\breve{N}=N$, $\hat{N}=\tilde{N}$. Now, we have the following cases

a) $\breve{N}\geq 2$, $\hat{N}\geq 2$
b) $\breve{N}=1$, $\hat{N}\geq 2$
c) $\breve{N}\geq 2$, $\hat{N}=1$
d) $\breve{N}=1$, $\hat{N}=1$
(a)–(d) and $\bar{D}_{\breve{N}}=\tilde{D}_{\hat{N}}$

e) $\breve{N}=1$, $\hat{N}\geq 2$
f) $\breve{N}\geq 2$, $\hat{N}=1$
g) $\breve{N}=1$, $\hat{N}=1$
(e)–(g) and $\bar{D}_{\breve{N}}\neq\tilde{D}_{\hat{N}}$

and in any case we can prolong

$$\ldots\{\ldots M(D_{\check N}[\check N],\tilde D_{\hat N}[\hat N])\}\ldots$$
$$\top$$
$$\ldots\{D_{\check N}[\check N](\tilde D_{\hat N}[\hat N])\}\ldots$$

Case g) means exactly that $j_1,\ldots,j_r$ is a solution of $\mathcal{L}$. The other cases express the contrary. Case g) leads to a node in $T_{\Pi_{\mathcal{L}}}$ which is not partially compilable:

$$\top$$
$$\ldots\{\tilde D_1[1](E,\varkappa_1\langle D_1\rangle,\varkappa_2\langle D_1\rangle,D)\}\ldots$$
$$\top$$
$$\ldots\{M(E,\bar E)\}\ldots$$
$$\top$$
$$\ldots\{E(\bar E)\}\ldots$$
$$\top$$
$$\ldots\{\bar E(E,D,D,M1)\}\ldots$$
$$\top$$
$$\ldots\{M1(E,\bar E)\}\ldots$$
$$\top$$
$$\ldots\{E(D)\}\ldots$$
$$\top$$
$$\ldots\{D(E,D,D,M1)\}\ldots$$

D is not applied appropriately. All the other cases lead to a maximal node with a dummy statement between the innermost call brace pair:

case a)

$$\top$$
$$\ldots\{\tilde D_{\hat N}[\hat N](D_{\check N-1}[N-1],\varkappa_1\langle D_{\check N}\rangle,\varkappa_2\langle D_{\check N}\rangle,D)\}\ldots$$
$$\top$$
$$\ldots\{D(D_{\check N-1}[\check N-1],\tilde D_{\hat N-1}[\hat N-1])\}\ldots$$
$$\top$$
$$\ldots\{\qquad\qquad\}\ldots$$

case b)

$$\top$$
$$\ldots\{\tilde D_{\hat N}[\hat N](E,\varkappa_1\langle D_1\rangle,\varkappa_2\langle D_1\rangle,D)\}\ldots$$
$$\top$$
$$\ldots\{D(E,\tilde D_{\hat N-1}[\hat N-1])\}\ldots$$
$$\top$$
$$\ldots\{\qquad\qquad\}\ldots$$

case c)

$$\top$$
$$\ldots\{\tilde{D}_1[1](D_{\check{N}-1}[\check{N}-1],\varkappa_1\langle D_{\check{N}}\rangle,\varkappa_2\langle D_{\check{N}}\rangle,D)\}\ldots$$
$$\top$$
$$\ldots\{D(D_{\check{N}-1}[\check{N}-1],\bar{E})\}\ldots$$
$$\top$$
$$\ldots\{\qquad\qquad\}\ldots$$

case d)

$$\top$$
$$\ldots\{\tilde{D}_1[1](E,\varkappa_1\langle D_1\rangle,\varkappa_2\langle D_1\rangle,D)\}\ldots$$
$$\top$$
$$\ldots\{D(E,\bar{E})\}\ldots$$
$$\top$$
$$\ldots\{\qquad\qquad\}\ldots$$

case e)

$$\top$$
$$\ldots\{\tilde{D}_{\hat{N}}[\hat{N}](E,\varkappa_1\langle D_1\rangle,\varkappa_2\langle D_1\rangle,D)\}\ldots$$
$$\top$$
$$\ldots\{M(E,\tilde{D}_{\hat{N}-1}[\hat{N}-1])\}\ldots$$
$$\top$$
$$\ldots\{E(\tilde{D}_{\hat{N}-1}[\hat{N}-1])\}\ldots$$
$$\top$$
$$\ldots\{\tilde{D}_{\hat{N}-1}[\hat{N}-1](E,D,D,M1)\}\ldots$$
$$\top$$
$$\ldots\{D(E,\begin{matrix}\bar{E} \text{ if } \hat{N}-1=1\\ \tilde{D}_{\hat{N}-2}[N-2] \text{ if } \hat{N}-1\geq 2\end{matrix})\}\ldots$$
$$\top$$
$$\ldots\{\qquad\qquad\}\ldots$$

case f)

$$\top$$
$$\ldots\{\tilde{D}_1[1](D_{\check{N}-1}[\check{N}-1],\varkappa_1\langle D_{\check{N}}\rangle,\varkappa_2\langle D_{\check{N}}\rangle,D)\}\ldots$$
$$\top$$
$$\ldots\{M(D_{\check{N}-1}[\check{N}-1|,\bar{E})\}\ldots$$
$$\top$$
$$\ldots\{D_{\check{N}-1}[\check{N}-1](\bar{E})\}\ldots$$
$$\top$$
$$\ldots\{\bar{E}(\begin{matrix}E \text{ if } \check{N}-1=1\\ D_{\check{N}-2}[\check{N}-2] \text{ if } \check{N}-1\geq 2\end{matrix},\varkappa_1\langle D_{\check{N}-1}\rangle,\varkappa_2\langle D_{\check{N}-1}\rangle,D)\}\ldots$$
$$\top$$
$$\ldots\{D(\begin{matrix}E\\ \text{or}\\ D_{\check{N}-2}[\check{N}-2]\end{matrix},\bar{E})\}\ldots$$
$$\top$$
$$\ldots\{\qquad\qquad\}\ldots$$

So we see that T_{Π} has a not partially compilable node if and only if there is a solution $j_1,\ldots,j_r$ of $\mathcal{L}$. Q.e.d.

For better illustration we construct $\Pi_{\mathcal{L}}$ for a concrete correspondence system $\mathcal{L}$ with

$$\Gamma = \{\gamma_1 = (c_1,\tilde{c}_1) = (BA,\overline{B}),\quad \gamma_2 = (c_2,\tilde{c}_2) = (B,\overline{A}\overline{B})\} \quad .$$

$\mathcal{L}$ has the solution 1,2 as

$$\overline{c_1}\,\overline{c_2} = \overline{BA}\,\overline{B} = \overline{B}\,\overline{A}\,\overline{B} = \tilde{c}_1\tilde{c}_2$$

(case g)) whereas e.g. 1 is no solution as

$$\overline{c_1} = \overline{BA} \neq \overline{B} = \tilde{c}_2$$

(case c)). The second part of $\Pi_{\mathcal{L}}$ looks as follows:

```
proc L1(x,y);
        {proc B[1](η);{η(x,D,M,D)};
         proc A[2](η);{η(B[1],M,D,D)};
         proc B̄[1](ξ,α,β,γ);{β(ξ,y)};
         L1(A[2],B̄[1]);L2(A[2],B̄[1]);M(A[2],B̄[1])};
proc L2(x,y);
        {proc B[1](η);{η(x,D,M,D)};
         proc Ā[1](ξ,α,β,γ);{α(ξ,y)};
         proc B̄[2](ξ,α,β,γ);{β(ξ,Ā[1])};
         L1(B[1],B̄[2]);L2(B[1],B̄[2]);M(B[1],B̄[2])
L1(E,E);L2(E,E) end
```

A subtree of $T_{\Pi_{\mathcal{L}}}$ is:

```
Π𝔏   = ...L₁(E,Ē)...

Π𝔏₁  = ...{proc B[1](η);{η(E,D,M,D)};
           proc A[2](η);{η(B[1],M,D,D)};
           proc B̄[1](ξ,α,β,γ);{β(ξ,E)};
           L₁(A[2],B̄[1]);L₂(A[2],B̄[1]);M(A[2],B̄[1])}...

Π𝔏₁₂ = ...{proc B[3](η);{η(A[2],D,M,D)};
           proc Ā[2](ξ,α,β,γ);{α(ξ,B̄[1])};
           proc B̄[3](ξ,α,β,γ);{β(ξ,Ā[2])};
           L₁(B[3|,B̄[3]);L₂(B[3],B̄[3]);M(B[3],B̄[3])}...

Π𝔏₁  = ...{..;M(A[2],B̄[1])}...        ...{B[3](B̄[3])}...
           ...{A[2](B̄[1])}...          ...{B̄[3](A[2],D,M,D)}...
           ...{B̄[1](B[1],M,D,D)}.      ...{M(A[2],Ā[2])}...
           ...{D(B[1],Ē)}...           ...{A[2](Ā[2])}...
           ...{          }...          ...{Ā[2](B[1],M,D,D)}...
           partially compilable        ...{M(B[1],B̄[1])}...
                                       ...{B[1](B̄[1])}...
                                       ...{B̄[1](E,D,M,D)}...
                                       ...{M(E,Ē)}...
                                       ...{E(Ē)}...
                                       ...{Ē(E,D,D,M1)}...
                                       ...{M1(E,Ē)}...
                                       ...{E(D)}...
                                       ...{D(E,D,D,M1)}...

                                       not partially compilable
```

The constructed programs $T_{\Pi_{\mathfrak{L}}}$ remain compilable in ALGOL60-SF if we add appropriate specifications. All formal parameters x, ξ get the specification <u>proc</u> (<u>proc</u>), y, η get <u>proc</u> (<u>proc</u>, <u>proc</u>, <u>proc</u>, <u>proc</u>), α, β, γ get <u>proc</u> (<u>proc</u>, <u>proc</u>). Lemma 8 remains true. The only difference in the proof is that in appropriate application of D reveals already in

...{E(D)}...

so that

...{D(E,D,D,M1)}...

cannot be a successor.

<u>Theorem 4:</u> It is undecidable whether a compilable ALGOL60-SF (or ALGOL60-PL/1) program Π has correct parameter transmission.

On the other hand, it is impossible to add appropriate specifications such that all $\Pi_{\mathfrak{L}}$ become compilable ALGOL60-68 programs and Lemma 8 remains true. If so then because of Lemma 2 there would not exist any solvable correspondence system. We can even prove directly for every Π :

<u>Lemma 9</u>: It is impossible to add appropriate specifications such that $\Pi_{\mathfrak{L}}$ becomes a compilable ALGOL60-68 program.

<u>Proof:</u> If the contrary would hold we had the following equations for declarator trees:

$$\partial E = \partial x^{L_1} = \partial C_{1 n_1}[n_1] = \partial x^M$$

$$\partial \bar{E} = \partial y^{L_1} = \partial \tilde{C}_{1 n_1}[\tilde{n}_1] = \partial y^M$$

As a consequence we can show up an equation

$$\underline{\text{proc}}\ (\partial\xi^{\bar{E}}, \partial\alpha^{E}, \partial\beta^{\bar{E}}, \partial\gamma^{\bar{E}}) = \underline{\text{proc}}\ (\partial x^D, \partial y^D)$$

as for the program Π^2. Contradiction! Q.e.d.

IX. Application of the Proof Methods on Other Problems

Theorem 5: It is undecidable whether two original programs are equivalent.

Corollary: There is no general algorithm which transforms any original program into an equivalent one without global formal parameters.

Proof of Theorem 5: For every $\mathcal{L}$ we construct two different programs $\Pi_{\mathcal{L}}^1$ and $\Pi_{\mathcal{L}}^2$. For $\Pi_{\mathcal{L}}^1$ the body {x(D)} of M1 in $\Pi_{\mathcal{L}}$ is replaced by {}, for $\Pi_{\mathcal{L}}^2$ by {**real** A;A:=A+1}. $\Pi_{\mathcal{L}}^1$ and $\Pi_{\mathcal{L}}^2$ are equivalent if and only if $\mathcal{L}$ has no solution. So equivalence of ALGOL 60-P programs is undecidable. This is true even for ALGOL 60-68 programs and consequently for ALGOL 60-SF and ALGOL 60-PL/1: For all formal parameters x,ξ we add the declarator tree $\underline{a}$, for y,η we add $\underline{b}$, for α,β,γ we add $\underline{c}$ with

$$\underline{a} = \underline{\text{proc}}\ (\underline{b}),$$
$$\underline{b} = \underline{\text{proc}}\ (\underline{a},\underline{c},\underline{c},\underline{c}),$$
$$\underline{c} = \underline{\text{proc}}\ (\underline{a},\underline{b}). \qquad \text{Q.e.d.}$$

It is impossible to prove Theorem 5 only with finite declarator trees as the following condition must hold:

$$\partial C_{1n_1}[n_1]=\underline{\text{proc}}(\partial C_{1,n_1-1}[n_1-1],\ldots,\ldots),\text{if } n_1\geq 2$$

or

$$C_{1n_1}[n_1]=\underline{\text{proc}}(\partial x^{L1},\ldots,\ldots,\ldots),\text{if } n_1=1.$$

In any case we have

$$\partial C_{1n_1}[n_1]=\underbrace{\underline{\text{proc}}(\ldots(\underline{\text{proc}}}_{\text{at least 1 time}}(\partial x^{L1},\ldots,\ldots,\ldots)\ldots)$$
$$=\underline{\text{proc}}(\ldots(\underline{\text{proc}}(\partial C_{1n_1}[n_1],\ldots,\ldots,\ldots)\ldots),$$

an equation which can only be fulfilled by infinite declarator trees. Therefore, we formulate the

Conjecture: Equivalence for ALGOL 60-68 programs becomes decidable if we restrict ourselves to programs with finite declarator trees.

Definition 11: A procedure φ in an original program Π is called formally reachable or simply reachable if there is a node Π' in T_{Π} whose innermost generated block is the modified body of a copy of φ.

Theorem 6: It is undecidable whether a procedure φ in an original program with correct parameter transmission is reachable.

Proof: M1 in $\Pi_{\mathcal{L}}^{2}$ is formally reachable if and only if $\mathcal{L}$ has a solution. This is true for all four languages. Q.e.d.

Definition 12: A procedure φ in an original program Π is called formally recursive or simply recursive if there is a path in T_{Π} with two different nodes whose innermost generated blocks are modified bodies of copies of φ. φ is called strongly recursive if there is a path in T_{Π} with two different nodes whose innermost generated blocks are modified bodies of identical copies of a copy of φ.

Theorem 7: It is undecidable whether a procedure in an original program with correct procedure parameter transmission is recursive resp. strongly recursive.

Proof: In $\Pi_{\mathcal{L}}^{2}$ we replace the body {real A; A:=A+1} by {M1(x,y)} and we get $\Pi_{\mathcal{L}}^{3}$. As M1 is a procedure, declared in the main program of $\Pi_{\mathcal{L}}^{3}$, M1 is recursive if and only if M1 is strongly recursive. M1 is recursive if and only if $\mathcal{L}$ has a solution. This is true for all four languages. Q.e.d.

Concerning Theorems 5 and 6 we have the analogous conjectures as the one formulated above.

By application of proof methods similar to those of Theorem 1 we may prove

Theorem 8: For programs without global formal procedure parameters it is decidable whether a procedure is reachable, recursive or strongly recursive.

X. Not Strongly Recursive Procedures

The difference between recursive and strongly recursive procedures is important for compilation techniques, because those procedures which are not strongly recursive allow a simpler implementation than others do. E.g. it is not necessary to reserve indexregisters for them. Fixed storage places for simple and auxiliary variables can be reserved among the fixed storage of the statically surrounding procedure so that we need an indexregister at most for this larger procedure.

If we conceive blocks as procedures without parameters called on the spot, then blocks are not strongly recursive. Loosely formulated: Not strongly recursive procedures can be handled like blocks.

Generated procedures, generated by the compiler as a substitute for complex expressions as actual parameters of procedure statements, may be recursive but are not strongly recursive. E.g.

```
...;P(A+B[2],X);...
```

has the compiled form

```
...;begin real proc G;{A+B[2]};P(G,X) end;...   .
```

See [3], page 119.

A for statement with two for list elements

```
for i:=A step B until C, Ā step B̄ until C̄ do S
```

is to be handled as if two for statements

```
for i:=A step B until C do S;
for i:=Ā step B̄ until C̄ do S
```

with identical controlled variables and controlled statements were given. Here the compiler will generate a procedure which is not strongly recursive

```
begin
  proc s; {S};
  for i:=A step B until C do s;
  for i:=A step B until C do s
end
```

The blocks and the generated procedures are not strongly recursive because of the following

Theorem 9: A procedure φ without parameters whose identifier f occurs in the main part only of the procedure body (or in the main program) in which the procedure is declared is not strongly recursive.

Proof: Let φ have the form

$$\underline{\text{proc}}\ f;\{\bar{\rho}\}$$

declared in the main part of the body of procedure ψ in Π:

$$\Pi = \ldots\ldots \underline{\text{proc}}\ g(x_1,\ldots,x_n);\underbrace{\{\ldots\underline{\text{proc}}\ f;\{\bar{\rho}\};\ldots\ldots\}}_{\bar{\rho}^{\psi}}\ldots\ldots$$

Let us have a look at a path in the execution tree T

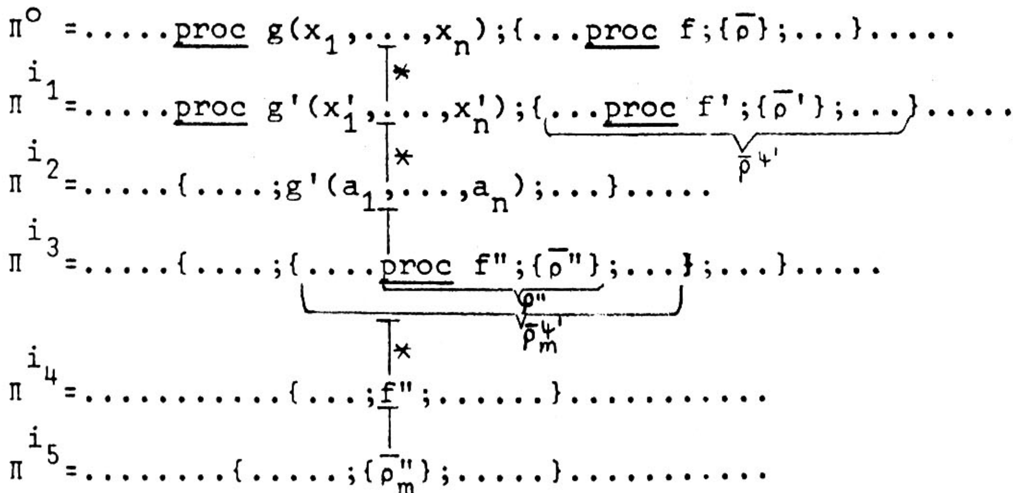
$$\Pi = \Pi^0 \vdash \Pi^1 \vdash \Pi^2 \vdash \ldots\ldots$$

We may assume that all programs Π^i are distinguished and that successive programs

$$\Pi^i \vdash \Pi^{i+1}$$

are literally identical with the exception of that procedure statement in Π^i which is replaced by a modified procedure body in order to give Π^{i+1}. So, every declaration δ in Π^i occurs also in Π^j for $j \geq i$ and there is a smallest number i_δ such that the declaration δ occurs in Π^{i_δ}. We call Π^{i_δ} the associated program of δ and of the identifier of δ.

Let the path have a node Π^{i_5} generated by a call of a copy φ'' of the procedure φ. Then, the path from Π^0 to Π^{i_5} has a structure



$$\Pi^0 = \ldots\ldots \underline{\text{proc}}\ g(x_1,\ldots,x_n);\{\ldots\underline{\text{proc}}\ f;\{\bar{\rho}\};\ldots\}\ldots\ldots$$

$$\Pi^{i_1} = \ldots\ldots \underline{\text{proc}}\ g'(x_1',\ldots,x_n');\underbrace{\{\ldots\underline{\text{proc}}\ f';\{\bar{\rho}'\};\ldots\}}_{\bar{\rho}^{\psi'}}\ldots\ldots$$

$$\Pi^{i_2} = \ldots\ldots\{\ldots\ldots;g'(a_1,\ldots,a_n);\ldots\}\ldots\ldots$$

$$\Pi^{i_3} = \ldots\ldots\{\ldots\ldots;\underbrace{\{\ldots\ldots\underbrace{\underline{\text{proc}}\ f'';\{\bar{\rho}''\}}_{\varphi''};\ldots\}}_{\bar{\rho}_m^{\psi'}};\ldots\}\ldots\ldots$$

$$\Pi^{i_4} = \ldots\ldots\ldots\ldots\{\ldots;\underline{f}'';\ldots\ldots\}\ldots\ldots\ldots\ldots$$

$$\Pi^{i_5} = \ldots\ldots\ldots\{\ldots\ldots;\{\bar{\rho}_m''\};\ldots\ldots\}\ldots\ldots\ldots\ldots$$

(Transitions $\Pi^0 \vdash^* \Pi^{i_1} \vdash^* \Pi^{i_2} \vdash \Pi^{i_3} \vdash^* \Pi^{i_4} \vdash \Pi^{i_5}$.)

Π^{i_3} is obviously the program associated to φ'' and to f". As φ'' has no parameters the associated program of any identifier h occurring in $\bar{\rho}''_m$ is Π^i with $i \leq i_3$ or $i \geq i_5$. Furthermore, h is different from f" by assumption.

Now, let $h(a'_1,\ldots,a'_{n'})$

be a procedure statement in the main part of $\bar{\rho}''_m$ and let Π^{i_5+1} result from Π^{i_5} by a call of $h(a'_1,\ldots,a'_{n'})$:

$$\Pi^{i_5} = \ldots\{\ldots;h(a'_1,\ldots,a'_{n'});\ldots\}\ldots$$
$$\Pi^{i_5+1} = \ldots\{\ldots;\{\bar{\rho}'''_m\};\ldots\}\ldots$$

Any identifier h' occurring in $\bar{\rho}'''_m$ has an associated program Π^i with $i \leq i_3$ or $i \geq i_5$, as we can easily see. Furtheron, h' cannot be equal to f". Otherwise, h would be the identifier of a procedure declared within $\bar{\rho}_m'$ parallel to φ'' and f" would occur in the body of h. This is impossible by assumption.

Iterating this argument we see that φ'' is never called a second time in the path. So φ is not strongly recursive. Q.e.d.

In literature it is sometimes proposed to handle the blocks and generated procedures as if they were general procedures. This simplifies the whole translation and interpretation process, in principle. In favour of efficient object code procedures which are not strongly recursive should be processed differently. Unfortunately, Theorem 6 says that there does not exist any general algorithm which figures out exactly these special procedures. Therefore, theorems like Theorem 9 have a great importance for compilation techniques.

XI. Concluding Remarks

In a certain sense ALGOL 60 programs with procedures may be considered to be a sort of macro grammars which have been studied in literature. In view of the results in |2|, Theorem 5 looks surprising, an observation of Dr.H.Feldmann which we have to thank him for. In a further paper on elimination of global parameters and on normal forms for programs with procedures we shall investigate similarities and differences between programs and macro grammars.

Literature

[1] *J.W. de Bakker and W.P. de Roever:* A Calculus for Recursive Program Schemes. MR 131/72, Mathematisch Centrum Amsterdam, February 1972.

[2] *M.J. Fischer:* Grammars with Macro-like Productions. Report No. NSF-22. Math. Ling. and Autom. Translation. Harvard Univ., Cambridge, Mass., May 1968.

[3] *A.A. Grau, U. Hill, H. Langmaack:* Translation of ALGOL 60. Handbook for Automatic Computation. Vol.I, Part b, Springer, Berlin-Heidelberg-New York 1967.

[4] *E.N. Hawkins and D.H.R. Huxtable:* A Multi-Pass Translation Scheme for ALGOL 60. Annual Review in Automatic Programming. Vol.III, pp. 163-205, Oxford, Pergamon Press 1963.

[5] *C.H.A. Koster:* On Infinite Modes. ALGOL Bulletin No. 30, pp. 86-89, February 1969.

[6] *P. Naur (Ed.) et al:* Revised Report on the Algorithmic Language ALGOL 60. Num. Math. 4, pp. 420-453, 1963.

[7] *C. Pair:* Concerning the Syntax of ALGOL 68. ALGOL Bulletin No. 31, pp. 16-27, March 1970.

[8] *H. Scheidig:* Representation and Equality of Modes. Inf. Proc.Letters 1, pp. 61-65, 1971.

[9] *A. van Wijngaarden, B.J. Mailloux, J.E.L. Peck, and C.H.A. Koster:* Report on the Algorithmic Language ALGOL 68. Num. Math. 14, pp. 79-218, 1969.

[10] *M. Zosel:* A Formal Grammar for the Representation of Modes and its Application to ALGOL 68. Univ. of Wash. 1971.

[11] *E.L. Post:* A Variant of a Recursively Undecidable Problem. Bull. Am. Math. Soc., Vol.52, pp. 264-268, 1946.