REVUE D'ANALYSE NUMÉRIQUE ET DE LA THÉORIE DE L'APPROXIMATION, Tome 3, N° 1, pp. 1974, 93–111

SOME OBSERVATIONS CONCERNING THE APPLICATION OF TREE THEORY TO THE SYSTEM GENERATOR

by

TEODOR RUS

(Cluj)

§. 1. Introduction

It is well-known that a System .Generator is a software which has the following task: given a software library (or a collection of software libraries) for a computer, it must generate a software system corresponding to both, a given configuration for that computer and the user options [1].

For this purpose it receives a series of infromations concerning the computer configuration, the libraries which must be processed, and the software components which must be included in the generated system. One of its most difficult operations is connected with the analysis of the components which must be included in the new software system. Generally, the structures of these components are given in a tree-form, using the program segmentation. At running-time the user must be free to choose a software configuration corresponding to his necessities.

In order to be able to preform this, some structure transformations of the trees representing the software components are required. So, the System Generator also receives informations concerning the structure of the components which must be included in the new software system. Now, there is the problem of answering the following questions:

1). Are the structures communicated to the System Generator proper to the functions which perform these components?

2) Are the structures communicated to the System Generator equivalent to those provided for by their constructor?

This paper is devoted to describing some algorithms which allow to answer these questions.

§. 2. The program segmentated structures

The solving of any questions by means of a computer needs to apply a determinated sequence of operations on the data of the questions. Generally, this sequence is called algorithm. Among these operations there are some which decide in what away the computing process will be continued. During the execution of a subsequence of operations, generally, other subsequence will have to wait.

For some large programs, the problem of performing different subsequences of the same algorithm on the same memory place arises. This leads us to the notion of segmentation. Formally, this means that the *program has a tree structure.*

The elements of this structure are the following:

1. There exists a sequence of a program called root, which also has the task to *manage the communication* between other sequences of the program. These sequences are called „of the first level". If any such sequence is performed in a given time, the others will wait for it.

2. Each first level sequence can be a root.

3. In a given program (1)—(2) can only be repeated finitely.

From (1)—(3) we can easily represent a program as a tree structure. More precisely, we shall define the following correspondence:

(a.1.) To every root in a program there corresponds a node in a tree.

(a.2.) To every first level of a program sequence there corresponds — in a given order — one node, connected with arrow to the root form (a.1.).

(a.3.)(a.1.)—(a.2.) is repeated until (1)—(3) are exhausted for a given program.

In this way, for an application of (a1)—(a2), we obtain a structure like the one of fig. 1, where R is a symbol noting the root, and $N1$, $N2$, ..., Nk are the symbols noting the first level sequence of the program.

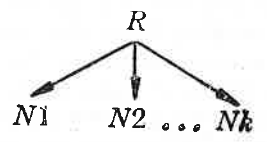


Fig. 1. Tree structure with one level

Now, if we are to repeat (a1)—(a2) we will obtain the tree structures like that described in [3].

Definition 2.1.: For a given program P we shall call the *program tree,* the tree which is generated by the correspondence (a1)—(a3).

In the next part of the paper the symbols noting the nodes of a program tree will be called segments and the arrows will be called edges.

For a faithful representation of a program, using the tree structures, we shall do the following transformation T of a program tree:

(T1) Transformation T transforms every node of the program tree into a segment, noted with symbol noting that node.

(T2) Transformation T transforms every edge of the program tree into a node attached to that segment which corresponds to the node from which this edge starts.

In this way, the tree of fig. 1 is transformed into the tree of fig. 2.

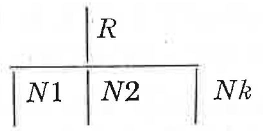


Fig. 2. Program structure

The obtained tree of a given program tree using transformation T will be called *program structure.*

Now we are going to define the notion of the execution of a program structure.

Definition 2.2.: *By the execution of a program P we mean the following process:*

F0. *The execution starts with the run of the sequence corresponding to the root of the program structure.*

F1. *The root of the program structure can call every segment belonging to it.*

F2. *The first level segments of a given program structure are performed in the following way:*

F2.1. *The root of the program structure can call every sequence corresponding to the first level segment of this root.*

F.2.2. *Any N_i segment already called can call every N_j, $i \neq j$ of the same level, and N_i will be covered by N_j.*

F2.3. *The communication between the N_i segments is made by means of their root.*

F.2.4. *The process starts with* F2.1.

F3. *The processes* F0—F2 *are the same for every subtree of a given program structure.*

Now, we shall define some useful notions in the following:

Definition 2.3.: (a) *The path of a program will be called a sequence of the segments corresponding to a path of the tree.*

(b) *A level of a program will be called the collection of all the segments corresponding to a level of the tree.*

(c) *A covering shall be a path beginning with the root of the structure and ending with a segment corresponding to an extremety of the program structure.*

If we note the *length* of a segment with LS, which is measured by the *number of memory words* needed for its loading, we get: the length of a path is the amount of the segment-lengths belonging to it. The length of the program is the length of the greatest covering of the program. Then, we suppose that every level of a program is *labelled* by the greatest of the paths beginning with the root of the structure and ending on this level.

§. 3. The reprezentation of the program structures

The segmentated structures described in Section 1 are the elements which must be dealt with by both, the programmer and the System Generator.

Generally, the programmers deal with these structures in the from of paranthesed expressions, and the System Generator deals with these structures in the form of lists.

The formal definition of a tree expression using BNF notations is as follows :

⟨SEGMENT⟩ : : = SYMBOL|SYMBOL EXPRESSION

⟨SEGMENTS LIST⟩ : := ⟨SEGMENT⟩|⟨SEGMENTS LIST⟩, ⟨SEGMENT⟩

TREE : : = ⟨SEGMENT⟩|⟨TREE⟩|(⟨SEGMENTS LIST⟩)|
⟨TREE⟩(⟨TREES LIST⟩)

⟨TREES LIST⟩ : : = ⟨TREE⟩|⟨TREES LIST⟩, ⟨TREE⟩

E x a m p l e s:

1. A
2. ALPHA (BETA, GAMMA, P1)
3. *P*1(*P*2(*P*3(*P*31, *P*32, *P*33), *P*4), *P*5)

The corresponding structures for the given examples are represented in fig. 3.

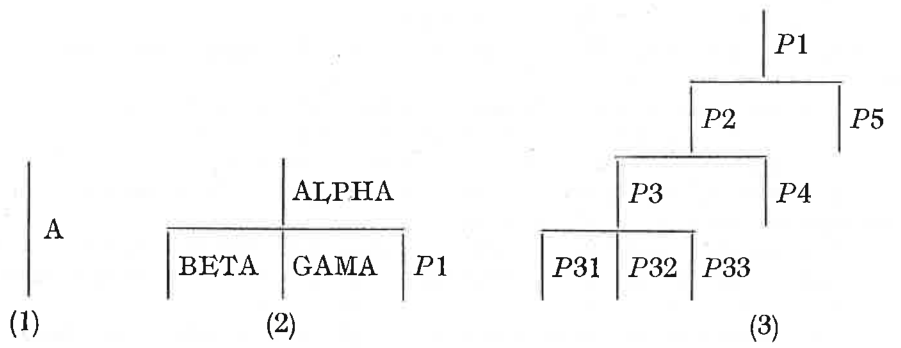


Fig. 3. Examples of program structures

O b s e r v a t i o n 3.1. : The notions of the symbol and those of symbol expression are the same as the notions of name or identifier and name expression, which are defined in programming languages.

Here the symbol expression is used for the structures which are connected with each other like both, that of the program segment and its corresponding data segment, and that of the segment and its corresponding common segments etc.

The lists corresponding to the tree expressions are defined according to the matricial representation of the trees [2]. In this way, to every program structure we attach a matrix with three columns, or equivalently with three ordered lists.

The first one of them is called NAME list.

The second list is called ADDRESS list and the third one is called CONNECTION .list.

The construction of these listst is made recursively. Now, we shall give the algorithms for the construction of these lists, having the tree expressions as data :

(1) The first step of the algorithm is to build the NAME list. For this purpose the algorithm works as follows :

N.1. Let an order be given for the program structure. In this order the first segment is to be the root of the structure. Then, the algorithm will put the sequence of the segments in a memory list in the given order, each being a record.

(2) The ADDRESS list is built in the following way :

A.1. The first record of the ADDRESS list will correspond to the root of the structure. It will be set on zero.

A.2. The order in this list is the same as in the NAME list. The segments on the first level of the structure have as records in the ADDRESS list the length of their root.

A.3. Let the records for all the segments on a given level be allready built. The greatest length of the paths, beginning with the root and ending on this level, is chosen as record for the next level.

(3) The CONNECTION list will be built in the following way :

Let $A_1, A_2, \ldots, A_n$ be all the coverings of the structure. The order defined in the NAME list will induce an order in the set of coverings. For this purpose let $A_1 < A_2 < \ldots < A_n$, each A_i being

$$A_i = \{S_{i1}, S_{i2}, \ldots, S_{in_i}\}$$

where all S_{ij} are the segments, and $S_{i1} < S_{i2} < \ldots < S_{in_i}$ in the NAME list order. Now, we get : $\bigcap_{i=1}^{n} A_i \neq \emptyset$ because each A_i begins with the root.

A record of the CONNECTION list consists of two elements. The first one is the order number of the NAME list to which the element corresponding to the current element of the CONNECTION list is connected.

The second element of the record depends on symbol expressions, if such a segment belongs to a symbol expression. Otherwise it is set on zero.

The order in this list is the same as that of the NAME list. The algorithm for the construction of this list is the following:

C.1. One builds the record for A_1 · If $A_1 = \{S_{11}, S_{12}, \ldots, S_{1n_1}\}$ then the record for A_1 will have 0, 1, 2, ..., n_1-1 as first elements.

C.2. Let $S_{2,k} \notin A_1 \cap A_2$ where S_{2k} is the first segment in the given order of $A_2 \setminus A_1 \cap A_2$. Then, the first corresponding element of its record will be n_k, i.e. the same as that corresponding to element S_{1k_1} on the same level, such as $S_{1k_1} \in A_1 \setminus A_1 \cap A_2$ is the first element in the given order, and $S_{1k_1} \notin A_1 \cap A_2$

For $S_{2k_1+1}, \ldots, S_{2n_2}$ the first element of the records will be the next natural numbers.

C.3. Let there exist a CONNECTION list for $A_1, A_2, \ldots, A_i$. then, for A_{i+1} we shall proceed as in C.2., where instead of A_1 we shall take A_i and insted of A_2 we shall take A_{i+1}.

C.4.C.3. will be repeated for $i = 1, 2, \ldots, n$.

E x a m p l e 1.: Let us take structure (3) of fig. 3. Then the NAME list, ADDRESS list, and CONNECTION list for this structure are represented in fig. 4.

Order	Pointer 1.
1	P1
2	P2
3	P3
4	P31
5	P32
6	P33
7	P4
8	P5

Order	Pointer 2.
1	L1 = 0
2	L2 = 1(P1)
3	L3 = 1(P1)+ 1(P2)
4	L4 = 1(P1)+ 1(P2)+ 1(P3)
5	L5 = L4
6	L6 = L4
7	L7 = L3
8	L8 = L2

Order	Pointer 3.	
1	0	0
2	1	0
3	2	0
4	3	0
5	3	0
6	3	0
7	2	0
8	1	0

Fig. 4. *The list representation of program structure 3, fig. 3.*

Here we have used the order: $P1 < P2 < P3 < P31 < P32 < P33 < P4 < P5$, and the coverings are:

$$A_1 = \{P1, P2, P3, P31\}$$
$$A_2 = \{P1, P2, P3, P32\}$$
$$A_3 = \{P1, P2, P3, P33\}$$
$$A_4 = \{P1, P2, P4\}$$
$$A_5 = \{P1, P5\}$$

where $A_1 < A_2 < A_3 < A_4 < A_5$.

The symbols POINTER 1, POINTER 2 and POINTER 3 are the adress pointers for the connection between the lists, and 1(P) is the length of program P.

§. 4. The equivalence transformations of the program structure

With a view to the purpose of modifying the run speed and memory allocation of a program we shall consider four types of transformations of the program structures.

The first one of them is the permutation operation of the subtree of a given tree on a given level. The second one is the dilatation operation of the program coverings. The third one of them is a condensation operation of program coverings, and the fourth one of them is the section operation of a subtree and its attaching to another subtree.

These operations will be noted by $\mathfrak{L}$, $\mathfrak{D}$, $\mathfrak{C}$, $\mathfrak{S}$ respectively, each of them being defined on a level i of the program structure.

Permutation operation: Let P be a given program structure. If $S_1, S_2, \ldots, S_p$ are its first level subtrees [3], then, according to the definition of tree expression

$$P = S_0(S_1, S_2, \ldots, S_p)$$

Now let $\Phi = (\Phi_1, \Phi_2, \ldots, \Phi_p)$ be a permutation of sequence $(1, 2, \ldots, p)$. Then, by definition we get:

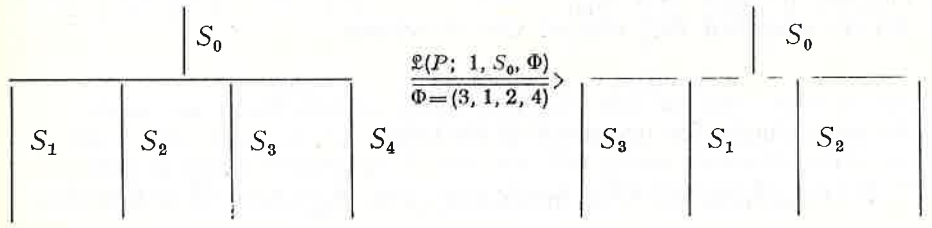
$$\mathfrak{L}(P;\ 1, S_0, \Phi) = S_0(S\Phi_1, S\Phi_2, \ldots, S\Phi_p)$$

If we consider the notion of execution defined by Definition 2.2., then, from this point of view we get:

$$P \sim \mathfrak{L}(P;\ 1, S_0, \Phi)$$

Operation $\mathfrak{L}$ can be defined recursively on each level of program structure P.

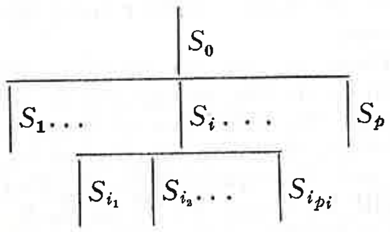
As an example of applying this operation we have



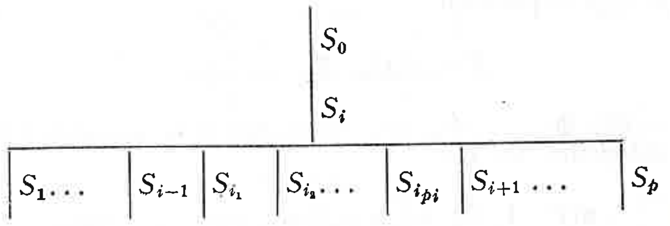
Dilatation operation: Let us again consider the structure $P = S_0(S_1, S_2, \ldots, S_p)$, and let us suppose that for a given i, $1 \leqslant i \leqslant p$ the subtree S_i can be represented by $S_i(S_{i1}, \ldots, S_{ip_i})$. Then, by definition on the first level subtree we get:

$$\mathfrak{D}(P\,;\ 1,\ S_i) = S_0(S_i(S_1,\ \ldots,\ S_{i-1},\ S_{i_1},\ \ldots,\ S_{ip_i},\ S_{i+1},\ \ldots,\ S_p))$$

i.e. a structure of the form:



after operation $\mathfrak{D}(P\,;\ 1,\ S_i)$ has been transformed into:



Condensation operation: This operation is, in a way, an inverse operation of $\mathfrak{D}$. Using the same notations for one $S_{ij,\,1<pi,}$ by definition we get:

$$\mathfrak{C}(P\,;\ 1,\ S_{ij}) = S_0(S_1,\ \ldots,\ S_{ij},\ S_i(S_{i_1},\ \ldots,\ S_{i_{j-1}},\ S_{i_{j+1}},\ \ldots S_{ip_i}) \ldots S_p)$$

Relation $P \sim \mathfrak{C}(P\,;\ 1,\ S_{ij})$ is true only if the following two conditions hold:

(i) S_{ij} does not transmit informations to S_i for the segments S_{i_1}, S_{i_2}, $\ldots$, $S_{i_{j-1}}$, $S_{i_{j+1}}$, $\ldots$, S_{ip_i}

(ii) S_{ij} does not call any of the structures

$$S_{i_1},\ S_{i_2},\ \ldots,\ S_{i_{j-1}},\ S_{i_{j+1}},\ \ldots,\ S_{ip_i}$$

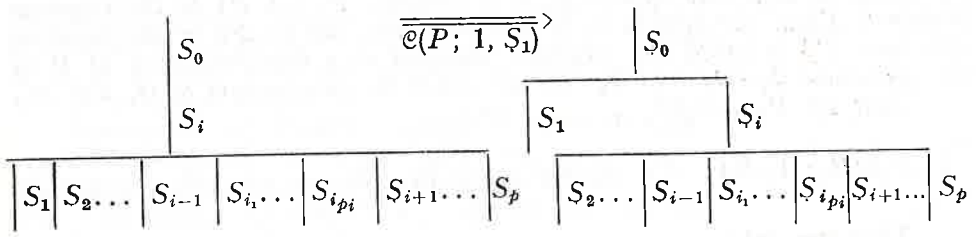
As an example for operation $\mathfrak{C}$ we take:

$$P = S_0(S_i(S_1,\ S_2,\ \ldots,\ S_{i-1}S_{i+1}S_{i_2},\ \ldots,\ S_{ip_i},\ S_{i+1},\ \ldots,\ S_p))$$

and

$$\mathfrak{C}(P\,;\ 1,\ S_1) = S_0(S_1,\ S_i(S_2,\ \ldots,\ S_{i-1}, S_{i_1}, S_{i_2},\ \ldots,\ S_{ip_i}, S_{i+1},\ \ldots,\ S_p))$$

or graphically



The condensation operation can be applied to any level of a given structure P if the following conditions hold:

— if the operation is of the form $\mathfrak{C}(P\,;\ i,\ S_k)$

and

if k *is the level* of S_k, then:

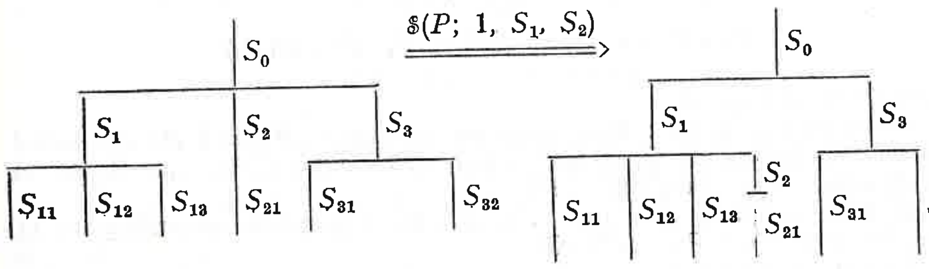
(1) $i \neq 0$
(2) $i < k$

Section operation: This is an operation for covering dilatation. It is denoted by $\mathfrak{S}(P\,;\ i,\ S_j,\ S_k)$ and it means that subtree S_k on level i will be sectioned and attached as first level structure to S_j on the same level i. If i = 1, as an example we get:

$$P = S_0(S_1,\ \ldots,\ S_i(S_{i_1},\ S_{i_2},\ \ldots,\ S_{ip_i}),\ \ldots,\ S_k,\ \ldots,\ S_p)$$

$$\mathfrak{S}(P\,;\ 1,\ S_i,\ S_k) = S_0(S_1,\ \ldots,\ S_i(S_{i_1},\ S_{i_2},\ \ldots,\ S_{ip_i}, S_k),\ \ldots,\ S_p)$$

or graphically: For $P = S_0(S_1(S_{11},\ S_{12},\ S_{13}), S_2(S_{21}),\ S_3(S_{31},\ S_{32}))$



Now, we shall explain more precisely the semnification of operation $\mathfrak{C}$. For this purpose let $\Omega = \{\mathfrak{T},\ \mathfrak{D}.\ \mathfrak{C},\ \mathfrak{S}\}$.

For a given structure P and for the given elements such as S_i, S_k (subtrees of P) on level i, and for a given permutation Φ, each $\mathfrak{T}$, $\mathfrak{D}$, $\mathfrak{C}$, $\mathfrak{S}$

will determine a concrete operation of the tpyes $\mathfrak{L}$, $\mathfrak{D}$, $\mathfrak{C}$, $\mathfrak{S}$. So, $\mathfrak{L}$, $\mathfrak{D}$, $\mathfrak{C}$, $\mathfrak{S}$ will be called operation schemes or operation types. $O_k \in \Omega$ is meant to be an operation of a given type.

D e f i n i t i o n 4.1.: *The composition of O_1, $O_2 \in \Omega$ is defined in the following way. One applies O_1 to a structure P. Let P1 be the resulting structure. Then, one applies O_2 to this structure, and let P2 be the resulting structure. P2 is called the structure obtained as a transformation of P by the operatione $O_2 \circ O_1$, and $O_2 \circ O_1$ are called the composition of O_1 and O_2.*

Now let $P_0 = S_0(S_1, \ldots, S_p)$ and

$$P = \mathfrak{D}(P_0 ; 1, S_i) ; \quad P = S_0(S_i(S_1, \ldots, S_{i-1}, S_{i_1}, \ldots, S_{i_{pi}}, S_{i+1}, \ldots, S_p))$$

Then we get:

$$\mathfrak{C}(P ; 1, S_1) = P_{01}$$

$$\mathfrak{C}(P_{01} ; 1, S_2) = P_{02}$$

$$\cdots\cdots\cdots\cdots$$

$$\mathfrak{C}(P_{0i-2} ; 1, S_{-1}) = P_{0i-1}$$

$$\mathfrak{C}(P_{0i-1}, 1, S_{i+1}) = P_{0i}$$

$$\cdots\cdots\cdots\cdots$$

$$C(P_{0p-1} ; 1, S_p) = P_{0p}$$

Now, there exists a permutation $\Phi = (\Phi 1, \Phi 2, \ldots, \Phi_p)$, such that $\mathfrak{L}(P_{0p} ; 1, S_0, \Phi) = P_0$

In other words, if P is obtained from a dilatation operation, then, there exist the operations $O_1, O_2, \ldots, O_{p-1}$ of type $\mathfrak{C}$, and O_p of type $\mathfrak{L}$, such that

$$O_p \circ O_{p-1} \circ \ldots \circ O_1(P) = P_0, \quad P = O(P_0)$$

where O is of type $\mathfrak{D}$.

D e f i n i t i o n 4.2.: *Two program structures P1 and P2 are called equivalent according to Ω, if there exist operations $O_1, O_2, \ldots, O_n$* of Ω *and $O_n \circ O_{n-1} \circ \ldots \circ O_1(P1) = P2$.*

If two program structures $P1$, $P2$ are equivalent according to Ω, then, we note that $P1 \simeq P2$ (Ω).

P r o p o s i t i o n 4.1.: $\simeq$ *is an equivalent relation.*

Proof: 1. $P1 \simeq P2$ $(\Omega) \Rightarrow P2 \simeq P1$ (Ω). This follows from the properties of the inversability of $\mathfrak{C}$ and $\mathfrak{D}$.

2. $P1 \simeq P2$ (Ω) and $P2 \simeq P3$ $(\Omega) \Rightarrow P1 \simeq P3$ (Ω) obviously.

3. $P \simeq P$ (Ω), if we consider the empty set of operations.

§. 5. An algorithm for the decidability of the equivalence structures

Supposing that $P1$, $P2$ are two program structures, the algorithm for the decidability, if either they are equivalent or not, tries to transform a structure, for istance, $P1$ into structure $P2$, using operations of the $\mathfrak{L}$, $\mathfrak{D}$, $\mathfrak{C}$, $\mathfrak{S}$ types. For this purpose we shall use two notions of equality of two program structures.

Therefore, let NAME1, ADDRESS1, CONNECTION1, respectively NAME2, ADDRESS2, CONNECTION2 characterize $P1$ and $P2$, and 1(NAMEi), 1(ADDRESSi), 1(CONNECTIONi), $i = 1, 2$ their lengths. By c(NAMEi), c(ADDRESSi), c(CONNECTIONi), $i = 1, 2$ we shall note the content of the lists.

D e f i n i t i o n 5.1.: *Two program structures P1 and P2 are called equal if:*

(i) 1(NAME1) = 1(NAME2)
1(ADDRESS1) = 1(ADDRESS2)
1(CONNECTION1) = 1(CONNECTION2)

(ii) c(NAME1) = c(NAME2)
c(ADDRESS1) = c(ADDRESS2)
c(CONNECTION1) = c(CONNECTION2)

and are noted by $P1 = P2$.

D e f i n i t i o n 5.2.: *Two structures P1 and P2 are called weakly equal* if

(i') the conditions of (i) *hold*

(ii'') c(ADDRESS1) = c(ADDRESS2)
c(CONNECTION1) = c(CONNECTION2)

and are noted by $P1 \underset{w}{=} P2$.

These two definitions are similar to that of [3].

D e f i n i t i o n 5.3.: *Two structures P1 and P2 are equal* (weakly equal) *until level i if the obtained program structures, by erasing the levels j, $j = i + 1, i + 2, \ldots$, are equal* (wea ly equal), *and are noted by*

$$P1 \overset{i}{=} P2, \quad \left(P1 \overset{i}{\underset{w}{=}} P2\right).$$

In the following we shall use the notion of equality until level i.

It is obvious that if $P1$, $P2$ are two program structures, and if their level numbers are $N1$, $N2$, then, they are equal (weakly equal) if they are equal (weakly equal) until level i, for $i = 0, 1, \ldots$, N, werhere $N = N1 = N2$.

The algorithm will be described on the levels in the following way:

Let P1 and P2 be two program structures, where P2 is considered to be an unchangable structure and P1 is transformed by the algorithm.

STEP 1: Let $i := 0$ be the level number. One verifies if the structures are equal until i, i.e. if they have the same root. If they are equal, then, the algorithm will continue with STEP 2; otherwise, they are not equivalent.

STEP 2: $i := i + 1$, and one verifies if $P1 \overset{i}{=} P2$. In order to be able to decide whether $P1 \overset{i}{=} P2$ we shall take all subtrees on level i for the structures $P1$ and $P2$. Let $k1$ and $k2$ be the numbers of these subtrees Only one of the following conditions holds:

(i) $k1 = k2$
(ii) $k1 < k2$
(iii) $k1 > k2$

(i) If $k1 = k2$, then, $P1 \overset{i}{\underset{w}{=}} P2$

Let $S_1^1, S_2^1, \ldots, S_p^1$ and $S_1^2, S_2^2, \ldots, S_p^2$, these subtrees having as roots S_i^1 respectively S_i^2, $i = 1, 2, \ldots, p$.

$$\text{If } \{S_1^1, S_2^1, \ldots, S_p^1\} = \{S_1^2, S_2^2, \ldots, S_p^2\}$$

in the sense of set theory, then, there exists a permutation

$$\Phi = (\Phi_1, \Phi_2, \ldots, \Phi_p) \text{ of } (1, 2, \ldots, p)$$

$$\text{such that} \qquad S'_{\Phi i} = S_i^2, \ i = 1, 2, \ldots, p,$$

and we get $\mathfrak{R}(P1; i, S_0^1, \Phi) = P_2$, where S_0^1 is the root of the first level structures $S_1^1, S_1^1, \ldots, S_p^1$.

Observation 5.1.: If $S_1^1, S_2^1, \ldots, S_p^1$ belongs to more than one root, then, there exist the segments $S_i^{\prime r_1}, S_i^{\prime r_2}, \ldots, S_i^{\prime r_j}$, which are roots for $S_1^1, S_2^1, \ldots, S_p^1$. Then, operation $\mathfrak{R}$ will be applied for corresponding permutations $\Phi^1, \Phi^2, \ldots, \Phi^j$. In this case we will get:

```
        k := 1
   ┌→ 𝔑(P1; i, S'ᵢᵏ, Φᵏ)
   │    k := k + 1
   └YES─ k ≤ j ─NO→
```

noted by $\prod_{k=1}^{i} \mathfrak{R}(P1;\ 1,\ S_i^k,\ \Phi^k)$
an algorithm which gives

$$\prod_{k=1}^{i} \mathfrak{R}(R1;\ i,\ S_i^{\prime k},\ \Phi^k) = P2$$

$$\text{If } \{S_1^1, S_2^1, \ldots, S_p^1\} \neq \{S_1^2, S_2^2, \ldots, S_p^2\}$$

in the sense of set theory, we can apply an equalizer procedure. This procedure is represented in Appendix *A*.

(ii) If $k1 < k2$ we shall look for a condensation operation on level i, because this operation will increase number $k1$. The operation will be determined in the following way:

Let $\mathcal{N}_{p_1}^i = \{S_1^1, S_2^1, \ldots, S_{k_1}^1\}$, $\mathcal{N}_{p_2} = \{S_1^2, S_2^2, \ldots, S_{k_2}^2\}$

(ii)$_1$ If $\mathcal{N}_{p_1}^i \not\subset \mathcal{N}_{p_2}^i$, then, the algorithm will be completed with the answer „nonequivalent structures".

(ii)$_2$ If $\mathcal{N}_{p_1}^i \subset \mathcal{N}_{p_2}^i$, then, the algorithms look for an operation of type $\mathfrak{C}$. This operation is built in the following way:

(ii)$_3$ One verifies if $|\mathcal{N}_{p_1}^i| = |\mathcal{N}_{p_2}^i|$.

If so, the algorithm will continue with STEP 2. Let S_e^2 be the first element of $\mathcal{N}_{p_2}^i$, such that $S_e^2 \notin \mathcal{N}_{p_1}^i$. One looks for S_e^2 among the first level structures of $S_1^1, S_2^1, \ldots, S_k^1$. If S_e^2 is found, then, the algorithm will continue. Otherwise, it will be completed with the answer „nonequivalent structures".

Let S_e^2 be among the first level structures of S_i^1, i.e. $S_i^1 = S_i^1(S_{i1}^1, \ldots, S_e^2, \ldots, S_{ip_i}^1)$. Then, the operation which will be applied is $\mathfrak{C}(P1; i, S_e^2)$. In this way we shall get $\mathcal{N}_{p_1}^i = \mathcal{N}_{p_1}^i \cup \{S_e^2\}$ and the algorithm will continue with (ii)$_3$.

(iii) If $k1 < k2$, the algorithm will look for an operation of dilatation or section on structure $P1$, because these operations decrease number $k1$. For determining this operation one again takes $\mathcal{N}_{p1}^i$ and $\mathcal{N}_{p2}^i$.

If $\mathcal{N}_{p2}^i \not\subset \mathcal{N}_{p1}^i$ the algorithm will be completed with the answer „nonequivalent structures".

If $\mathcal{N}_{p2}^i \subset \mathcal{N}_{p1}^i$, then, we may obtain:

(iii)$_1$ $k_1 \geqslant 2$, $k_2 = 1$. This case determines a dilatation operation. For this purpose let us suppose that $\mathcal{N}_{p2}^i = \{S_1^2\}$ and $\mathcal{N}_{p1}^i = \{S_1^1, S_2^1, \ldots, S_{k1}^1\}$, then, we must get $S_1^2 = S_i^1$ for one $i \in \{1, 2, \ldots, k_1\}$. The operation wlli be $\mathfrak{D}(P1; i, S_i^1)$, and the algorithm will continue with STEP 2.

(iii)$_2$ $k_2 > 1$, then, the algorithm looks for an operation of section. For this purpose it will proceed as follows:

(iii)$_3$ One verifies if $|\mathcal{N}_{p_1}^i| = |\mathcal{N}_{p_2}^i|$. If so, the algorithm will continue with STEP 2. Otherwise, the algorithm will continue with *(iii)*$_4$.

(iii)$_4$ Let S_e^1 be the first element of $\mathcal{N}_{p1}^i$ with $S_e^1 \notin \mathcal{N}_{p_2}^i$ In this case we may obtain the following:

(iii)$_5$. S_e^1 is a first level subtree of one of $S_1^2, S_2^2, \ldots, S_{k2}^2$ Let S_j^2 be this subtree. Then we shall apply $\mathfrak{S}(P1; i, S_j^2, S_e^1)$, and the algorithm will continue with (iii)$_3$.

(iii)$_6$ S_e^1 is not among the first level subtree of $S_1^2, S_2^2, \ldots, S_{k2}^2$. Then, let S_i^1 be the first subtree of $\mathcal{N}_{p1}^i$ equal to the first subtree S_j^2 of $\mathcal{N}_{p2}^i$. One applies $\mathfrak{D}(P1; i, S_i^1)$ and $i := i - 1$. The algorithm will continue with STEP 2.

This algorithm will be repeated for $i = 0, 1, \ldots, N2$. If from $P1$ we have obtained a structure for $i = N2$, which is equal to $P2$, then, we get

$$P1 \simeq P2 \ (\Omega)$$

otherwise, $P1$ is not equivalent with $P2$.

Observation 5.2.: For the System Generator $P2$ represents the structure which is communicated by the user, and P1 represents the structure defined in the system. In this way the System Generator will only include the software component $P2$ in the new software system if it is equivalent to $P1$.

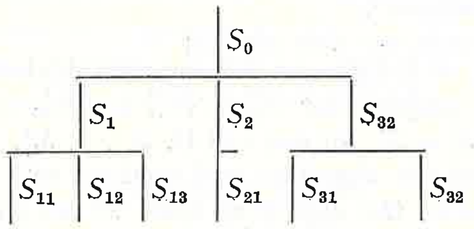
Observation 5.3.: The algorithm described above will work on the lists NAME, ADDRESS and CONNECTION. For this purpose it will be accompanied by an initial step, STEP 0, which transforms a tree expression in the NAME list, ADDRESS list, and CONNECTION list.

§. 6. Application

In this section we will deal with an example concerning the application of the algorithm, in order to make it easier for us to understand the latter. We shall apply the algorithm graphically and in the form of lists of the program structures.

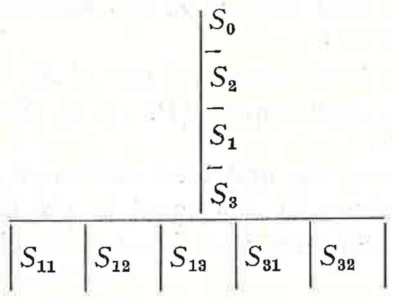
For this purpose let a software component be of the form:

$P_1 =$



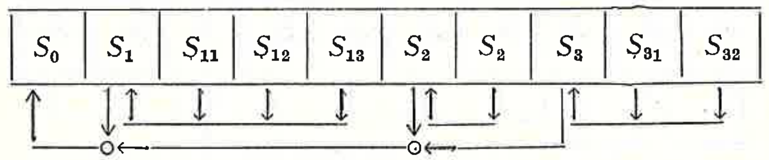
A software system must be generated, where this component has the structure $S0(S2(S1(S3(S11, (S12, S13, S21, S31, S32))))$ or, in other words, a program structure of the form:

$P_2 =$

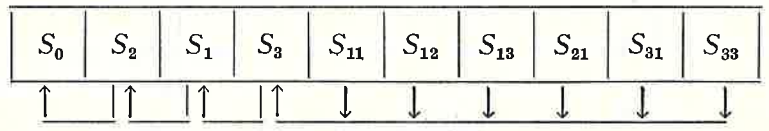


The list structures for these two structures will be represented in a short form as follows:

For the first structure we have:



and for the second one we have:



For applying the algorithm we have
$N(P1) = 3$, $N(P2) = 4$

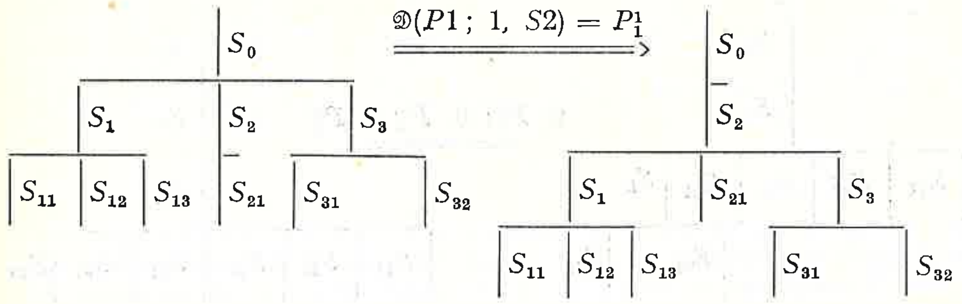
For $i := 0$, $P1 \overset{o}{\underset{w}{=}} P2$ because S_0 is the common root

For i = i + 1 one applies step (i) of the algorithm, and we obtain $k_1 = 3$, $k_2 = 1$, and the next step of the algorithm is (iii), because $k_1 <$ $< k_2$, $k_2 = 1$. Therefore,

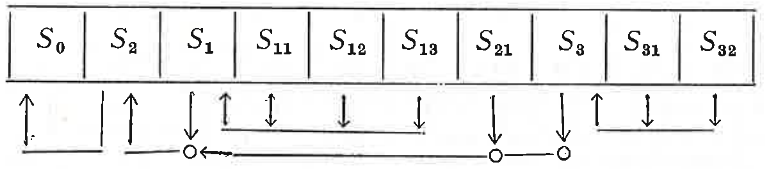
$$\mathfrak{N}^1_{p_1} = \{S_1, S_2, S_3\}, \ \mathfrak{N}^1_{p_2} = \{S_2\}, \ \mathfrak{N}^1_{p_2} \subset \mathfrak{N}^1_{p_1}$$

$$S_1 \in \mathfrak{N}^1_{p_1}, \ S_1 \notin \mathfrak{N}^1_{p_2},$$

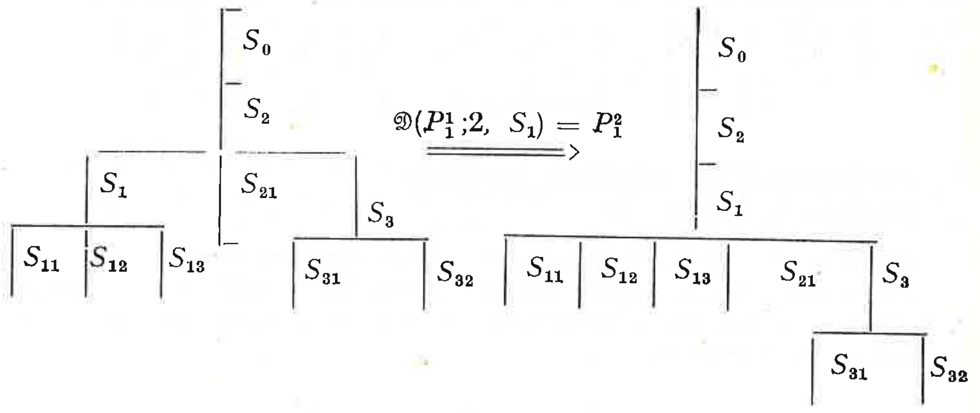
S_1 is not a first level subtree of S_2. A dilatation operation will be applied, according to $(iii)_6$.



Performing $i := i - 1$ we again get $i = 0$, and the algorithm will continue from (i). Now $i := i + 1$, respectively $i = 1$, and the new structure of the lists is:

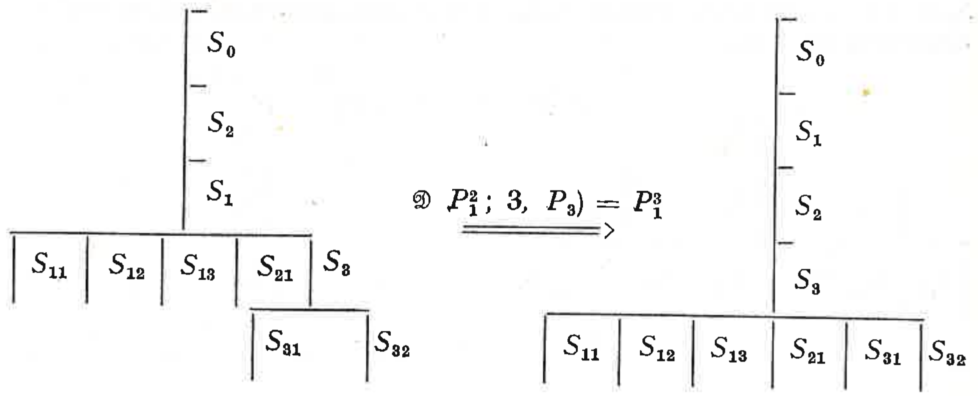

S_0 S_2 S_1 S_{11} S_{12} S_{13} S_{21} S_3 S_{31} S_{32}

So, we have $\mathfrak{D}(P1; 1, S_2) = P2$, and for $i = i + 1$, $i = 2$, according to the algorithm, a new dilatation operation will be applied.


S_0
S_2
S_1
S_{21}
S_3
S_{11} S_{12} S_{13}
S_{31} S_{32}
$\mathfrak{D}(P_1^1; 2, S_1) = P_1^2$

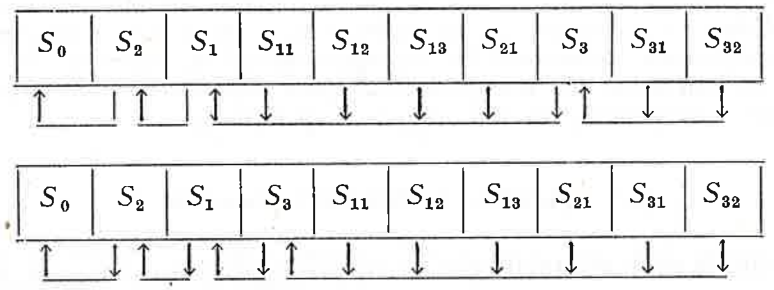
and $\mathfrak{D}(P_1^1; 2, S_1) = P2$.

Being the same case, $k_1 < k_2$, $k_2 = 1$, a new dilatation operation will be applied:

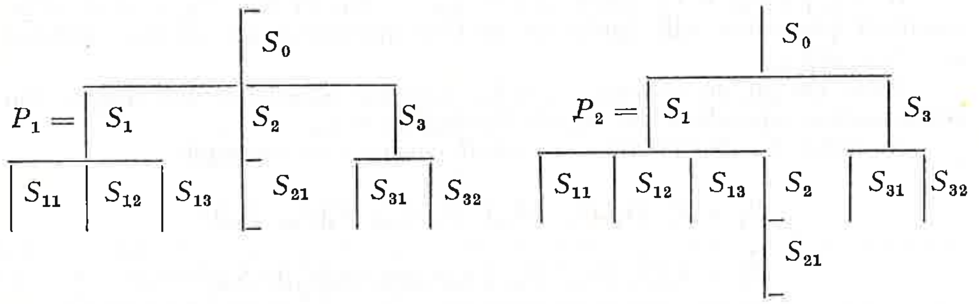

S_0
S_2
S_1
S_3
S_{11} S_{12} S_{13} S_{21}
S_{31} S_{32}
$\mathfrak{D}\ P_1^2;\ 3,\ P_3) = P_1^3$

And we can see now that $\mathfrak{D}(P_1^2; 3, S_3) = P2$, i.e. $P1 \approx P2$ (Ω).

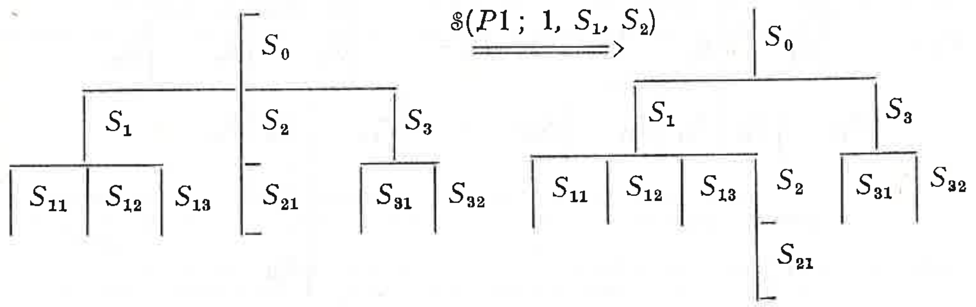
The successive structures of lists are:


S_0 S_2 S_1 S_{11} S_{12} S_{13} S_{21} S_3 S_{31} S_{32}
S_0 S_2 S_1 S_3 S_{11} S_{12} S_{13} S_{21} S_{31} S_{32}

Example 2: Let us consider a case, where a section operation is applied, such as:



The algorithm works like above, but in case (iii) we shall apply a section operation, because $k_1 > k_2$


$\mathfrak{S}(P1;\ 1,\ S_1,\ S_2)$
S_0
S_1 S_2 S_3
S_{11} S_{12} S_{13} S_{21} S_{31} S_{32}

Observation 6.1.: The operations of the types $\mathfrak{D}$ and $\mathfrak{S}$ are not independent. It is easy to show that any operation of type $\mathfrak{S}$ can be expressed by a composition of an operation of the types $\mathfrak{D}$ and $\mathfrak{C}$.

Any operation of type $\mathfrak{D}$ can be expressed by a composition of an operation of the types $\mathfrak{S}$ and $\mathfrak{C}$, too.

They both have been considered in order to simplify the algorithm.

O b s e r v a t i o n 6.2.: *Having in view the order supplied by the list representation of the structures, the connection in the lists can easily be established by a function on the subtrees of a given tree. Considering the order modification determined by the operation of the types $\mathfrak{A}$, $\mathfrak{D}$, $\mathfrak{C}$, $\mathfrak{S}$, it is easy to determine the transformation from one structure of lists to the next one corresponding to the respective operations.*

§.7. Appendix A. Equalizer procedure

In this section we shall give an equalizer procedure for case $k_1 = k_2$. For this purpose let $\mathfrak{N}^i_{P_1} = \{S^1_1, S^1_2, \ldots, S^1_p\}$ $\mathfrak{N}^i_{P_2} = \{S^2_1, S^2_2, \ldots, S^2_p\}$.

If $\mathfrak{N}^i_{P_1} \cap \mathfrak{N}^i_{P_2} = \emptyset$, then the algorithm is completed with the answer „nonequivalent structure".

If $\mathfrak{N}^i_{P_1} \cap \mathfrak{N}^i_{P_2} \neq \emptyset$, then let $\mathfrak{N}^i_{P_1, P_2} = \mathfrak{N}^i_{P_1} \cap \mathfrak{N}^i_{P_2}$, $\mathfrak{N}_{P_1, P_2} \neq \emptyset$. The equalizer procedure will apply the section operations for all the subtrees of $\mathfrak{N}^i_{P_1} \setminus \mathfrak{N}^i_{P_1 P_2}$.

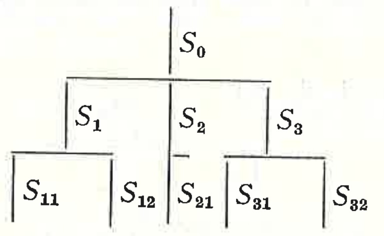
Thus we go on to case $k_1 < k_2$, and the algorithm will apply the condensation operation for again having $k_1 = k_2$.

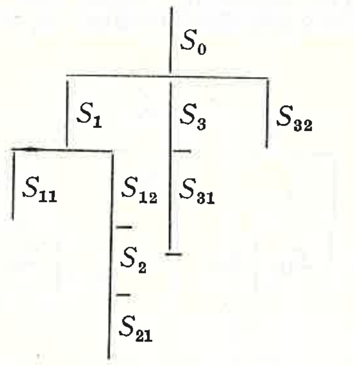
In order to clarify this, we shall consider an example:

$$P_1 = S_0\ (S_1\ (S_{11},\ S_{12}),\ S_2(S_{21}),\ S_3(S_{31},\ S_{32}))$$

$$P_2 = S_0(S_1\ (S_{11},\ S_{12}\ (S_2(S_{21}))),\ S_3(S_{31}),\ S_{32})$$

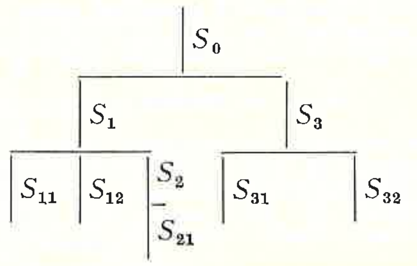
or graphically:

$P_1 =$ 

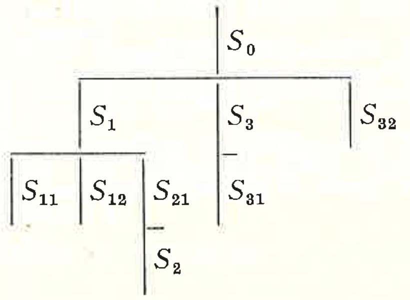
$P_2 =$ 

Now, $\mathfrak{N}^1_{P_1} = \{S_1, S_2, S_3\}$, $\mathfrak{N}^1_{P_2} = \{S_1, S_3, S_{32}\}$, $\mathfrak{N}^1_{P_1, P_2} = \mathfrak{N}^1_{P_1} \cap \mathfrak{N}^1_{P_2}$, $\mathfrak{N}^1_{P_1, P_2} = \{S_1, S_3\}$, $\mathfrak{N}^1_{P_1} \setminus \mathfrak{N}^1_{P_1, P_2} = \{S_2\}$

The section operation will be $\mathfrak{S}(P1\,;\ 1,\ S_1,\ S_2)$, and the structure obtained is:

$P'_1 =$ 

Now, according to the algorithm, for $k_1 < k_2$ we shall apply $\mathfrak{C}(P'_1\,;\ 1,\ S_{32})$, and the new structure is:



and we get: $k_1 = k_2$ on the first level of the given structures. In this way the algorithm can continue.

O b s e r v a t i o n 7.1.: *The equalizer procedure can be applied iff in the segmentation of the program the conditions (i) and (ii) of the definition of condensation have been respected. This observation holds for every case, where a condensation operation is applied. The condensation operation can only be applied without any restrictions if an operation of dilatation or section on the same level on the same segment has been applied.*

R E F E R E N C E S

[1] *Générateur de Système.* Manuel de Presentation, CII, Paris (1970).

[2] R u s, T., *On the matricial representation of the trees.* Mathematica, **6 (29)**, 2, 327–334 (1964).

[3] R u s, T., *Some observations concerning the application of the electronic computers in order to solve nonarithmetical problems.* Mathematica, **9 (32)**, 2, 334–360 (1967).

Received 12. IV. 1973

Institutul de tehnică de Calcul – Cluj.