Marek Perkowski

SYMBOLIC ANALYSIS OF (SEQUENTIAL AND PARALLEL) PROGRAM SCHEMATA IN THE DIGITAL DESIGN AUTOMATION SYSTEM

1. INTRODUCTION

Much effort has gone in recent years into program verification [Naur 66] , [Birm 74] , automatic invariants synthesis [Wegb 74] , [Ders 77] , symbolic simulation [King 76] , program optimization [Glus 70] , [Katz 78] , equivalence proving [Ersh 73] and other problems related to the semantics of programs [Ersh 73] , [Glus 70] . Some methods have been also elaborated for microprograms [Kime 75] , [Leem 75] . The only paper on hardware verification that I have come across is that by T. Wagner [Wagn 77] .

A new, interesting approach to the semantics of algorithms is developed in the papers of A. Mazurkiewicz and A. Blikle [Blik 72] , [Blik 73] , [Mazu 74] , [Budk 78] . The theory contained there (the MB - theory) can constitute a basis for practical (both hand and automatic) methods of analysis of sequential programs. As we will present in the sequel, this theory can be also extended to include parallel programs and can contribute to their optimization. We will also present how the methods can be programmed and utilized as a part of the system for automatic design of MOS LSI custom design circuits [Perk 79 a] .

It is assumed, that the reader is acquinted with the notions of the MB - theory; we hope however that this is not indispensable and that, together with the description, the presented examples will be self-explaining.

For a more detailed presentation of the notions, methods, programs, and also for more examples, the reader is referred to [Perk 79 b] .

The distinguishing features of the methods presented in this paper are the following:

- they can be used to both: parallel and sequential program schemata (microprograms, automata). In the parallel case, however, the synchronized execution is assumed,
- they can be applied to programs with register transfers and full jumps appearing on the "hardware level", rather than to those programs which are structural, recursive and equipped with other "high - level" constructs,
- it is possible to evaluate resultant relations, states of computation, and invariants, without any help from the user, contrary to other approaches in which the user has to specify some invariants [Wegb 74] , [Ders 77] , [Lewi 79] ,
- the finite, first term expansions of the above data can be evaluated in the most complicated cases; the advantage of our approach, however, lies in the fact that both types of the evaluation can be freely intermixed,
- the possibility of "symbolic traces" of states enables the analysis of control programs and automata, which operate in real - time and respond with sequences of output signals to sequences of input signals .
- the methods can be also used in program optimization, equivalence proving (in the sense of equivalent resultant relations or equivalent traces for some variables), in symbolic simulation, estimation of complexity, and in discovering time conflicts between viariables,

- consideration of some hardware properties by the applications mentioned above is possible.

Apart from the differences to other methods presented above, our approach differs from that of T. Wagner [Wagn 77] in the following aspects: we use the procedural language instead of a nonprocedural one and deal with the higher, algorithmic level of design. We are then interested in linking microprogram verification with hardware verification, rather than in verifying hazards, races and other similar phenomena. In the entire process of design such phenomena are discovered in our system on the lower level of description by use of the traditional simulation. Because our symbolic analysis package is a part of the greater design system, we tend to avoid the hazard detection as often as possible, by supplying the sophisticated design algorithms themselves so that they are able to produce hazardless and race-less circuits.

2. SYMBOLIC ANALYSIS OF SEQUENTIAL PROGRAMS

The basic foundations and some general information on the system for automatic design of digital devices (interchangeably - systems, automata) are given in [Perk 79 a] and [Perk 79 c]; more details are contained in [Perk 78] and [Perk 79 b]. This system, called DIADES (DIgital Automata DESigner) is implemented at the Institute of Automatic - Control, Technical University of Warsaw. In the system the algorithm of the behaviour of the digital system under design is given to the computer in ADL language and then compiled to the internal representation in the form of relational structures in language GRAF. The last form of the description is most useful as the input to the optimization and symbolic analysis programs.

At first we will introduce some notions and notation used below. The value of (COND $(p_i \ r_i)$ $(p_j \ r_j)$ ...) is r_i if predicate p_i is fulfilled, r_j if p_j is fulfilled (and not $\neg p_i \wedge p_j$ as in LISP), and so on. T meanes true and F - false. When value of the predicate is treated as the value of the arithmetic varitable - T is identified with 1 and F (NIL) with 0 . NAT is the set of natural numbers. The symbol * is a counterpart of "don't care" in logic design. It means that the value of the corresponding variable or predicate is not specified (or not of interest) at a given moment.

The variables can be classified according to different criteria, they are of the following types: output, input and internal; parallel, serial, pulse and logical (one - bit); combinational (not stored) and stored (stored variables are implemented as "general registers" like counters, shift registers, memories, switches, etc.). Bits from i - th to j - th of variable X ("i" is of less weight, it starts from 0) are denoted by X[i # j]. H is a symbol of the value "disconnection". It is applied to parallel output variables, connections with data - bus, etc.. Symbol H means that the value of corresponding variable is not transferred at a given moment or that the unit implementing the variable is disconnected (of such value is, for instance, a parallel, output variable V of a device in all moments of time other than those when the statement : V*- ⟨expression⟩ is executed). For logical variables value 0 is taken instead of H .

The external system (object) is defined as a certain system (digital or analog and digital) the inputs of which include

a set of output variables of the system under design and the outputs of which include a set of inputs of the system under design. The fixed block is a block (unit) which according to the ADL program declaration, is not to be changed by the transforming programs. We assume that the n - bit counter is the modulo 2^n counter if its capacity has not been declared separately. The arithmetical operations are modulo the capacity of storing register (for instance addition in counter is modulo its capacity).

Concatenation is denoted by ' ; a^n is defined: $a^1 :: = a$, $a^{n+1} :: = a^n , a$. Concatenation is used to create time strings of values of variables (in this paper referred to as "symbolic traces"). For instance if A is a boolean variable then $A^n , A^m , * A^k , A$ denotes time string of values of variable A : variable A is equal to 1 for n pulses of CLOCK, then it is equal to 0 for m pulses, then it is not specified (not of interest) for k pulses and next equal to 1 for one pulse. The string can be also described as $A = 1^n , 0^m , *^k , 1$. CLOCK is the main clock of a device - it will serve as a "scale of time" in all further time-oriented considerations.

In order to discover various mutual dependences among internal variables, as well as among input and output signals of the device - the user has a possibility of determining many forms and patterns of tracing of collection of variables. For instance, if A and B are input signals and C is the output signal, then we can trace separately according to the pattern (A , B , C) ; we can trace the input signal A together

with B (as a "state of inputs") and C separately - the pattern ((A, B), C); other examples of patterns:
((A, B, C)); ((A, C), B); (A, (C, B)), etc..
If A, B, and C are signals from Fig. 1 then we have:

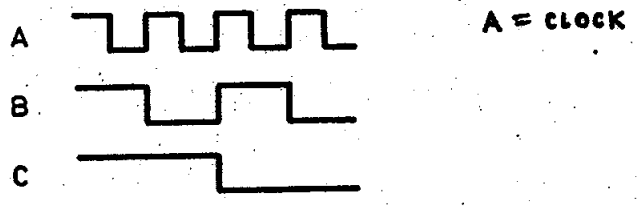


Fig. 1

$(A, B, C) \Rightarrow ((1,0)^4, (1^2, 0^2)^2, 1^4, 0^4)$,
$((A, B), C) \Rightarrow (((1, 1)'(0, 1)'(1, 0)'(0, 0))^2, 1^4, 0^4)$,
$((A, B, C)) \Rightarrow ((1, 1, 1)'(0, 1, 1)'(1, 0, 1)'\ldots (0,0,0))$.

Parallel variables can be traced in two forms, detalized, and non-detalized. In the detalized form each "state of variable" in the sequence is given with the precision of one clock pulse. In the non-detalized description the subsequent states of a variable are taken only for those moments in which its value can change. The variables for the symbolic trace, their forms and patterns are each time specified by the user.

The state of the variable X in a trace X^n can be considered as continuous (when the combinational (==) or asynchronous (**) modes of assignments are used in register - transfer statements) or as interrupted; according to the clock pulses (in the case of synchronous assignments (*=) changing the state with the arrival of the front slope of the pulse).

In the proposed approach the relations and symbolic states of the computation have wider meaning than in the MB - theory. They correspond not only to relations between data and results of the programs or to the symbolic states of program variables,

but also to relations between the input and output sequences of variables (the symbolic traces (detalized or non-detalized) are used in such a case).

Symbolic analysis is applied here for several purposes, the first is the program verification, i.e. an attempt to state whether the analysed program behaves in accordance with the designer's assumptions, i.e. whether the program fulfils some (detalized or non-detalized) time relations between sequences of values of variables, relations between input and output vectors of variables, or invariants.

The computation of resultant relations, states of computation and invariants is carried out by program CERTIF, which makes use of several rules and axioms for simplification. These are of two types: general and specific. General ones are used for all analysed programs and include Boolean, arithmetic, and "hardware" rules and axioms. The specific ones are declared by the designer for each specific design problem. They can be of the same types as the general ones.

The user can specify some input, output and input/output traces as impossible. In such a case the system answers with consequences of such assumptions. For instance, when we specify input sequence A as impossible, we quarantee that such sequence cannot occur at input - some output sequences can be thus simplified. If we specify a certain output sequence as impossible, we ask - "what will be the consequence in input sequences" - i.e. what input sequences are dangerous or not desired.

The resultant data can be also applied for equivalence proving and optimization. Equivalence can be based on:

EC 1) equivalent resultant relations of two programs,

EC 2) equivalent (some or all) symbolic states of computation

for the same admissible initial vector of variables,

EC 3) equivalent detalized traces,

EC 4) equivalent non-detalized traces.

Two programs can be equivalent with respect to one criterion and not with respect to the other. For instance if two programs answer with the detalized traces, correspondingly:
$B^{2n+1}, H^{m-1}, \neg B^{n}, H^{n}$ and $B^{n+1}, H^{m-n}, B^{n}, \neg B^{n}, H^{2n-1}$ for the input sequence $A^{2n}, \neg A^{m}, A^{2n}$ then they are equivalent with respect to EC 4 (because they have the same non-detalized trace $B^{2n+1}, \neg B^{n}$) while are not equivalent with respect to EC 3.

The following stages of the analysis of sequential programs are distinguished in our approach:

1) Create relational graph RG corresponding to the program-graph. RG has nodes Si corresponding to states of computation and arrows Ri corresponding to primitive transient relations between states. Each relation has a form $Ri = [Q_i]$ or $[p_i \mid Q_i]$ where p_i is a certain transition predicate and $Q_i = [\langle \text{vector of variables} \rangle := \langle \text{vector of assignment statements} \rangle]$ is a multiple assignment. The vectors of variables in states and relations considered below include not only stored variables, but also combinational or logic (control) variables and predicates corresponding to signals from units, indicators, etc. Some coordinates can be added formally to the vector as a means of measuring time and they do not find their counterparts in the hardware. Not all conditional signals must be included into the vector, but only those which are specified by the user.
2) Compile from RG a set of equations describing states and relations between them. Solve this set by applying: certain

rules of elimination of variables, the rule for solving fix-point equation $X = AX \cup B$, and rules for simplification of regular expressions. The resulting generalized regular expression describes all sets of finite and infinite (i.e. non-halting) paths in the graph.

3. Make special simplifications of the expression taking into account the specific semantics for the analysed program - graph. The rules for the nested (e.g. $((X^* Y)^* \ldots)$), subsequent (e.g. $X^* Y^* Z^* \ldots$), and (most often) correlative loops (e.g. $(X^* \cup Y^* \cup \ldots)^*$) are applied. For instance $(R1 \cup R2)(R1 \cup R2)^*$ can be replaced by $(R1^n R2)^m$ for some loops (n,m - natural numbers). Then substitute relations for symbols of relations in the regular expression. When only the resultant input/output relations between vectors of variables, or only non-detalized relations for sequences of variables are to be computed - make the simplifications which "forget" the detalization of traces to single pulses, by transforming the traces into the resultant relations.

4. Generate conditions of non-halting in infinite parts of the expression by evaluating the corresponding relations.

5. If the equation generated as the last by solving the set of equations has the form $S = AS$ then the program is cyclic (this is often the case in a certain class of control automata). The resultant relation is A^∞ and symbolic state $A^\infty S_0$. Instead, generate data A^M and $A^M S_0$.

6. Evaluate the resultant relation, (optionally) other transient relations, symbolic states and invariants. If the " traced variables " declared by the user are to be traced in detalized form, use the nonsimplified relation from p. 3. In the course of evaluating and subsequent simplifying the resultant rela-

tion use special rules and symbolic manipulation. The values of internal and output variables are unequivocally determined at each moment of the computation by the previous state of computation; while the values of input variables can change at any moment and thus cannot be computed according to the program from other current values of variables. The values of such variables can be however partially specified, by considering the subsequent stage of control. For instance, if: $R = [\, p \mid Q \,]$ is the primitive transient relation from state SI to state SJ and $p = \neg A \wedge B$ (where A is an input variable), and the automaton has come from SI to SJ, then in state SI the value of A was 0. If $p = \neg A \vee B$ then in state SI : A = 0 for the value B = 0 in SI and A = * for B = 1.

Because of the limited scope of the paper, we are not able to give here a full list of notions of MB - theory and DIADES references. We feel therefore that it will be more useful to study some examples in greater detail. In doing so the reader has to bear in mind that programs under discussion here generate AND - OR trees of possible solutions in the simplifying or optimizing search. Only some paths in these trees will be shown, mainly those which have proved successfull or most fruitful post factum. The example which follows (as well as the others) is the simplified version derived from a practical engineering problem.

Example 1

Problem formulation

The subsystem of a system of central data registration is to be designed. This subsystem has to process 64 analog data into the digital form and to check whether their values do not exceed their lower and upper limits.

The external system, which includes multiple word multiplexer and analog/digital converter, is shown on Fig. 2 (A/D converter has register B at its output).

The system under design has a fixed block - read only memory (ROM). The upper and lower limits of each processed data are stored in this memory as shown on Fig. 3. If:

lower limit [I] < X [I] ⩽ upper limit [I]

then the system goes on to the next data, else an alarm is printed on a line printer (number of data). A/D processing is performed in one clock pulse.

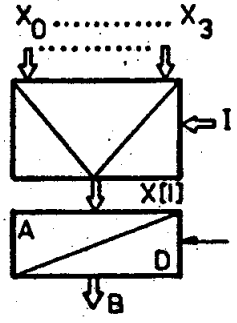


Fig. 2

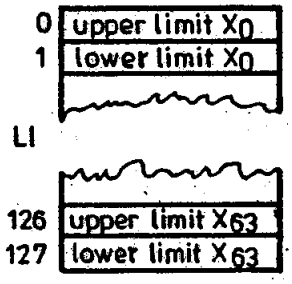


Fig. 3

The process of the symbolic analysis

The program of the subsystem behaviour is written by the designer in the ADL language in accordance with the given assumptions. This program is compiled into GRAF language, with program - graph **PO** corresponding to the flowdiagram from Fig. 4a as the result.

I is the counter of the processed data, LI is the address counter for ROM, OUT is the output parallel variable with the alarm number given to the line printer subsystem, AL is the alarm signal (logic variable) to the flip-flop on the panel. P1 is the predicate equal T iff (B>Y[LI]); it is logic variable- output from the unit - "comparator of arithmetic order > ", which has B and Y[LI] variables as inputs. Y[LI] is the subscripted variable coresponding to ROM. I, LI, B and Y are stored variables, OUT is the combinational variable, AL and P1 are logical variables. B is the input variable, AL and OUT - output variables, the rest of variables is internal. Relational graph created from P0 is shown on Fig. 4b. It is assumed, that in the initial moment the control automaton is in the state S1 and that in this moment LI = I = 0. The designer wishes variables OUT and AL to be "traced" (these variables will be underlined on the left sides of the relations).

The meaning of the relations in the relational graph is as follows:

R1 = [(B,I,LI,$\underline{OUT}$,$\underline{AL}$,P1):=(X[I], I, LI, H, 0, *)],

R2 = [¬(B>Y[LI]) | (B,I,LI,$\underline{OUT}$,$\underline{AL}$,P1):=(B,I,LI+1, H, 0, *)],

R3 = [(B>Y[LI]) (B,I,LI,$\underline{OUT}$,$\underline{AL}$,P1):=(B,I,LI,I,1,1)],

R4 = [(B,I,LI,$\underline{OUT}$,$\underline{AL}$,P1):=(B,I,LI+1, H, 0, *)],

R5 = [¬(B>Y[LI]) | (B,I,LI,$\underline{OUT}$,$\underline{AL}$,P1):=(B,I,LI,I,1,0)],

R6 = [(B>Y[LI]) | (B,I,LI,$\underline{OUT}$,$\underline{AL}$,P1):=(B,I+1, LI+1,H,0, *)],

R7 = [(B,I,LI,$\underline{OUT}$,$\underline{AL}$,P1):=(B,I+1, LI+1, H,0, *)].

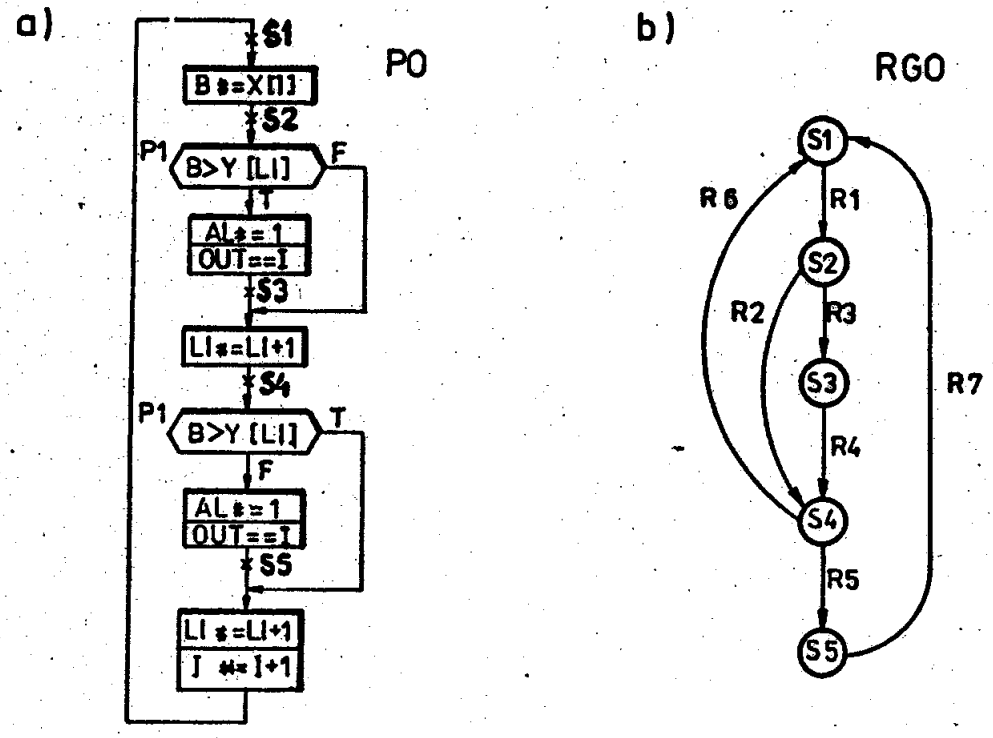


Fig. 4.

At first we will investigate the correctness of the P0 program. On the basis of the relational graph from Fig.4 b the program CERTIF compiles the following set of equations:

1) $S1 = R1\ S2$,

2) $S2 = R2\ S4 \cup R3\ S3$,

3) $S3 = R4\ S4$,

4) $S4 = R5\ S5 \cup R6\ S1$,

5) $S5 = R7\ S1$.

Reducing S3 and S5 the following equations replace equations 2), 3), 4) and 5) :

2') $S2 = (R2 \cup R3\ R4\ )\ S4$,

4') $S4 = (R6 \cup R5\ R7\ )\ S1$.

Substituting 4') into 2') :

2") $S2 = (R2 \cup R3\ R4\ )\ (\ R6 \cup R5\ R7\ )\ S1$.

Then

1') $S1 = R1\ (R2 \cup R3\ R4\ )\ (R6 \cup R5\ R7\ )\ S1$.

From theorem 3 from [Blik 73] CERTIF program evaluates:

$S1 = R^{\infty}\ S1_0 = [R1\ (R2 \cup R3\ R4\ )\ (R6 \cup R5\ R7\ )]^{\infty}\ S1_0$.

($S1_0$ is the initial state of symbolic state S1).

Next CERTIF evaluates the resultant relation, and other relations (optionally). We will now consider the evaluation of the resultant relation.

$R3\ R4 = [(B > Y[LI]\ ) \mid (B,I,LI,\underline{OUT},\underline{AL},P1) := (B,I,LI + 1\ ,$
$I\ '\ H,1\ '\ 0\ ,\ \ast\ )$;

$R2 \cup R3\ R4 = [(B,I,LI,\underline{OUT},\underline{AL},P1\ ) := (B,\ I,\ LI + 1,$
$(COND\ ((\neg (B > Y\ [\ LI\ ]))\ \ H\ )(\ (\ B > Y[LI]\ )\ (I\ '\ H))),$
$(COND\ ((\neg (B > Y\ [\ LI\ ]))\ \ 0\ )(\ (\ B > Y[LI]\ )\ (1\ '\ 0))),\ \ast\ ]$;

$R1\ (R2 \cup R3\ R4\ ) = [(B,\ I,\ LI,\ \underline{OUT},\ \underline{AL},\ P1) := (X[I]\ ,\ I\ ,\ LI + 1,$
$(COND\ ((\neg (X[I] > Y[LI]))\ \ H^2\ )\ (\ (\ X[I] > Y[LI]\ )(H'I'H))),$
$(COND\ ((\neg (X[I] > Y[LI]))\ \ 0^2\ )\ (\ (\ X[I] > Y[LI]\ )(\ 0'1'0\ ))),\ \ast\ )]$;

$R5\ R7 = [\neg (B > Y\ [LI]\ ) \mid (B,\ I,\ LI,\ \underline{OUT},\underline{AL},\ P1) := (B,\ I+1;\ LI+1,$
$I\ '\ H,1\ '\ 0\ ,\ \ast\ )]$;

$R6 \cup R5\ R7 = [(B,\ I,\ LI,\ \underline{OUT},\ \underline{AL},\ P1) := (B,\ I+1,\ LI + 1,$
$(COND\ ((B > Y[LI]\ )\ \ H\ )\ (\ (\neg (B > Y[LI]))\ (I\ '\ H))),$
$(COND\ ((B > Y[LI]\ )\ \ 0\ )\ (\ (\neg (B > Y[LI]))\ (1\ '\ 0)),\ \ast\ )]$.

$R=R1(R2 \cup R3R4)(R6 \cup R5R7)= [(B,I,LI,OUT,AL,P1):=(X[I],I+1,(LI+1)+1,$
$(COND((\neg(X[I]>Y[LI]))(COND((X[I]>Y[LI+1]) H^3)((\neg(X[I]>Y[LI+1]))$
$(H^2,I,H))))((X[I]>Y[LI])(COND((X[I]>Y[LI+1])$
$(H,I,H^2))((\neg(X[I]>Y[LI+1]))(H,I,H,I,H)))),$
$(COND((\neg(X[I]>Y[LI]))(COND((X[I]>Y[LI+1]) O^3)$
$((\neg(X[I]>Y[LI+1]))(O^2,1,0)))$
$((X[I]>Y[LI])(COND((X[I]>Y[LI+1])(0,1,O^2))$
$((\neg(X[I]>Y[LI+1]))(0,1,0,1,0))))),*)].$

This relation describes detalized traced variables with the accuracy of one pulse. It can be then utilized in two ways. In order to evaluate the resultant relation alone (possibly with non-detalized traces) symbols H are deleted. The "always applicable" reducing transformations are next performed. The relation obtains the form:

$R=[(B,I,LI,OUT,AL,P1):= (X[I], I+1, LI+2, (COND(((\neg(X[I]>Y[LI]))\wedge$
$(X[I]>Y[LI+1]))OUT)((((\neg(X[I]>Y[LI]))\wedge(\neg(X[I]>Y[LI+1])))$
$\vee((X[I]>Y[LI])\wedge(X[I]>Y[LI+1])))(OUT,I))$
$(((X[I]>Y[LI])\wedge(\neg(X[I]>Y[LI+1])))(OUT,I,I))),$
$(COND(((\neg(X[I]>Y[LI]))\wedge(X[I]>Y[LI+1]))AL)((((\neg(X[I]>Y[LI]))$
$\wedge(\neg(X[I]>Y[LI+1])))\vee((X[I]>Y[LI])\wedge(X[I]>Y[LI+1])))$
$(AL,1))(((X[I]>Y[LI])\wedge(\neg(X[I]>Y[LI+1])))(AL,1,1))),*)].$

CERTIF evaluates also symbolic states of computation for stored and control variables, and the invariants for all states of the relational graph. Assuming the vector of variables (B, I, OUT, AL, P1) and initial state $Si_0 = (*, 0, 0, *, *, *)$ it gets:

$Si_M = R^M Si_0 = (X[(M-1)_{mod64}], M_{mod64}, (2 \times M)_{mod128},$
CONCAT OUT, CONCAT AL,
$(COND((X[(M-1)_{mod\,64}] > Y[(2 \times M)_{mod\,128}]) T)$
$((\neg(X[(M-1)_{mod\,64}] > Y[(2 \times M)_{mod\,128}]))\ F))$

where CONCAT OUT and CONCAT AL are unsimplified symbolic traces of variables OUT and AL. The invariant is also generated:

$(B = X[(I-1)_{mod\ 64}]) \wedge (LI = (2 \times I)_{mod\ 128}) \wedge \ldots$

Similarly other symbolic states of the variables, and invariants for other states of the relational graph are generated.

Now CERTIF formulates the goal of reducing the resultant relation (optionally, the reduction can be also performed for other items) in accordance to the preprogrammed general simplification rules and general axioms, as well as the specific axioms and rules, specified each time by the designer.

The first goal G1 is to simplify the predicate:

$(\neg(X[I] > Y[LI])) \wedge (X[I] > Y[LI+1]))$.

This leads to goals G2 and G3 (G1 is the OR-node):

$(Y[LI+1] > Y[LI])$, $(Y[LI+1] = Y[LI])$

in order to match one of the following simplification rules:

$\neg(?A > ?B) \wedge (?A > ?C) \wedge (?C > ?B) \longrightarrow F$ (R1)

or

$\neg(?A > ?B) \wedge (?A > ?C) \wedge (?C = ?B) \longrightarrow F$ (R2)

(By ?X we will denote in rules the variable X).

Neither goal G2 nor G3 is fulfilled, and the assertions $(Y[LI+1] > Y[LI])$ and $(Y[LI+1] = Y[LI])$ are marked in the data-base as false for further use . CERTIF now tries to achieve goal G4 : to simplify the predicate :

$((\neg(X[I] > Y[LI])) \wedge \neg(X[I] > Y[LI+1]))$
$\vee((X[I] > Y[LI]) \wedge (X[I] > Y[LI+1]))$

It is first decomposed into three independent goals (G4 is the AND-node): G5 , G6 and G7. G5 is to simplify:

$(\neg(X[I] > Y[LI]) \wedge \neg(X[I] > Y[LI+1]))$ to ?U

G6 to simplify:

$((X[I] > Y[LI]) \wedge (X[I] > Y[LI+1]))$ to ?V

and G7 to simplify the resultant predicate (?U ∨ ?V).

Goal G5 would match one of the simplification rules:

$\neg(?A > ?B) \wedge \neg(?A > ?C) \wedge (?B > C) \rightarrow (?A > ?C)$ (R3)

$\neg(?A > ?B) \wedge \neg(?A > ?C) \wedge (?B = C) \rightarrow (?A > ?C)$ (R4)

if one of the goals G8: $(Y[LI] > Y[LI+1])$ or

G9: $(Y[LI] = Y[LI+1])$ were fulfilled (G5, just as the rest of nodes described below, is the OR-node).

Goal G8 is then called, which according to the specific axiom:

$(\forall ?LI \in NAT_{mod\ 128})[EVENP(?LI) \Rightarrow (Y[?LI] > Y[?LI+1])]$ (AS1)

leads to the new goal G10: EVENP (?LI). Now, according to the general axiom:

$(\forall ?X)[(\exists ?Y)[?X = 2 \times ?Y] \Rightarrow EVENP(?X)]$ (A1)

the goal G11 : (LI = 2 x ?Y) is called in the context for Si_M. Because (LI = 2 x M) in Si_M the goal G11 is fulfilled with ?Y = M. Facts F1 = (EVENP(LI)) and F2 = $(Y[LI] > Y[LI+1])$ are then asserted as true in the data-base. The simplification rule R3 for G5 is then applied, which leads to the predicate:

$(\neg(X[I] > Y[LI+1]))$.

Similarly subgoal G6 leads to subgoals G12:

$(X[LI] > Y[LI+1])$ and G13 : $(X[LI] = Y[LI+1])$.

Goal G12 is now found directly in the data-base, and next the simplification rule:

$(?A > ?B) \wedge (?A > ?C) \wedge (?B > ?C) \rightarrow (?A > ?B)$ (R5)

is applied to G6, which leads to the predicate:

$(X[I] > Y[LI])$.

The program turns back to G7 to simplify the predicate, which has resulted from applications of G5 and G6. The new goal G7 is

then to simplify:

$(\neg (X[I] > Y[LI+1]) \vee (X[I] > Y[LI]))$.

This leads to goals G14: $(Y[LI+1] = Y[LI])$,

(to satisfy the rule (R6) $[\neg(?A > ?B) \vee (?A > ?C)] \wedge (?B = ?C) \rightarrow T)$,

G15: $(X[I] > Y[LI+1])$ and G16: $(\neg (X[I] > Y[LI]))$.

Goal G14 is found as asserted false and goals G15 and G16 are disproved.

To simplify the predicate

$((X[I] > Y[LI]) \wedge \neg (X[I] > Y[LI+1]))$

once more rule R1 and assertion F2 are used, which simplifies the above predicate to F.

Substituting the reduced predicates to R and taking into account that $[F \mid Ri] = NIL$, relation R is next reduced to the form:

$R = [(B, I, LI, \underline{OUT, AL,} P1) := (X[I],\ I+1,\ LI+2,$

(COND $(((\neg (X[I] > Y[LI])) \wedge (X[I] > Y[LI+1]))$

OUT)

$(((\neg (X[I] > Y[LI+1])) \vee (X[I] > Y[LI]))$

$(OUT\ 'I)))$,

(COND $(((\neg (X[I] > Y[LI])) \wedge (X[I] > Y[LI+1]))$

AL)

$(((\neg (X[I] > Y[LI+1])) \vee (X[I] > Y[LI]))$

$(AL\ '1)))$, $\star)]$.

The relation R^M and $S1_M$ are evaluated:

$S1_M = R^M S1_o = [X[(M-1)_{mod64}],\ M_{mod64},\ (2 \times M)_{mod\ 128},$

CONCAT OUT, CONCAT AL, CONCAT P1]

where CONCAT OUT is printed in the form:

CONCAT

IF ((X[0] > Y[0]) v (X[0] ≤ Y[1])) THEN 0 ELSE H

IF ((X[1] > Y[2]) v (X[1] ≤ Y[3])) THEN 1 ELSE H

IF ((X[2] > Y[4]) v (X[2] ≤ Y[5])) THEN 2 ELSE H

. . .

IF (($X[(M-1)_{\text{mod } 64}] > Y[(2 \times M - 2)_{\text{mod } 128}]$)
v ($X[(M-1)_{\text{mod } 128}] \leq Y[(2 \times M - 1)_{\text{mod } 128}]$))

THEN $(M-1)_{\text{mod } 128}$ ELSE H

which means that the simplified symbolic trace of output variable OUT is equal to the string with length M, whose first element is 0, when (X[0] > Y[0]) v (X[0] ≤ Y[1]) holds or otherwise it has the value "not connected"; next element is 1, when, ..., etc.

The first-term expansion of CONCAT OUT is for M = 2 the following tree:

```
0   ((X[0] > Y[0]) v (X[0] ≤ Y[1]))
I
I  ->T  1   ((X[1] > Y[2]) v (X[1] < Y[3]))
I       I
I       I  ->T (0 , 1)
I       I
I       L  -> F (0)
L->F    1   ((X[1] > Y[2]) v (X[1] ≤ Y[3]))
        I
        I -> T (1)
        I
        L -> F ( )
```

First row describes predicate with number 0 and the second the predicate with number 1. The tree means that if predicates 0 and 1 are both fulfilled, the simplified symbolic trace of OUT

is (0 , 1). The traces of variables can be also obtained in detailed form with the accuracy of one pulse. In such a case we must start from the initial relation R , without simplifying symbols H.

On the basis of the printed data the designer decides whether the ADL program given to the computer was adequate to his wishes, i.e. whether he agrees with the printed consequences of the program. We see that in this case the resultant data show that the program PO fulfils the assumptions of the designer.

3. ANALYSIS OF PARALLEL PROGRAM SCHEMATA

The source language ADL of DIADES makes it possible to describe programs with parallel branches. The language includes several aids to describe parallelism, including the possibility of description of Petri nets and some other models of parallel processing. We will describe statements: FORK, COND, DEXOR and DAND.

FORK is the description of the program-graph node in which control divides itself into parallel branches. When branches outpointing from COND node correspond to nondisjoint predicates, COND divides control to all branches to which lead predicates fulfilled at the moment. Now the control in branches can be executed independently, if there is no synchronization. DROP is the statement of control termination in one of the parallel branches. Each DROP stops execution of one FORK branch, while the controls in other branches go on. DAND and DEXOR serve to describe nodes in which controls join together, coming from parallel branches. DAND describes a node which is passed by the control when all controls from inpointing branches have reached it; DEXOR - a node which transfers the control even when only one control from the inpointing branches

has reached the node. In such a case the controls are removed from all other inpointing branches.

A program is considered parallel when it has at least one FORK statement or COND statement with nondisjoint predicates, otherwise it is sequential. We assume that both types of programs are implemented as synchronous automata (special units implement DEXOR and DAND statements). It can be proved that there exists a constructive method to find an equivalent sequential relational graph RG (as the graphs from the previous section) for each parallel program [Perk 79 b].

The symbolic analysis of the parallel program can be therefore reduced to the method outlined in section 2. The entire process comprises the following stages:

- transformation of the parallel program-graph into the parallel relational graph (PRG) with non-disjoint relations outpointing from the graph nodes,
- transformation of the parallel relational graph into the sequential relational graph,
- calculation of transient relations between nodes in the new graph,
- calculation of the resultant relation of the program (according to the method from section 2).

If in a graph arrow from node n to node m has description R, than n is a predecessor, and m is a R-successor of n. This is denoted also by $S(n, R) = m$.

For computing transient relations in RG from relations in PRG we will introduce operation Ω, defined as follows.

Let: $R_1 = [p_1 \mid (x_1, x_2, \ldots, x_n) := (f_1, f_2, \ldots, f_n)]$,

$R_2 = [p_2 \mid (x_1, x_2, \ldots, x_n) := (g_1, g_2, \ldots, g_n)]$.

Then

$R_1 \Omega R_2 = [p_1 \wedge p_2 \mid (x_1, x_2, \ldots, x_n) := (h(f_1, g_1), h(f_2, g_2), \ldots, h(f_n, g_n))]$.

Function h can be defined in many ways, depending on hardware realization of transfers between registers. We will define here the following function:

$$h(f_i,g_i) = \begin{cases} g_i \text{ - when } f_i = x_i \text{ ,} \\ f_i \text{ - when } (f_i \neq x_i) \wedge \left[(g_i = x_i) \vee (g_i \neq x_i) \wedge (f_i = g_i)\right], \\ \text{error "conflict of variable } x_i\text{"} \text{ - in other cases.} \end{cases}$$

The algorithm to transform the parallel relational graph into the equivalent sequential one is given in [Perk 77] and [Perk 79 b]. As an example consider PRG from Fig.5a with one FORK and one DEXOR node. The corresponding SRG is shown on Fig. 5 b.

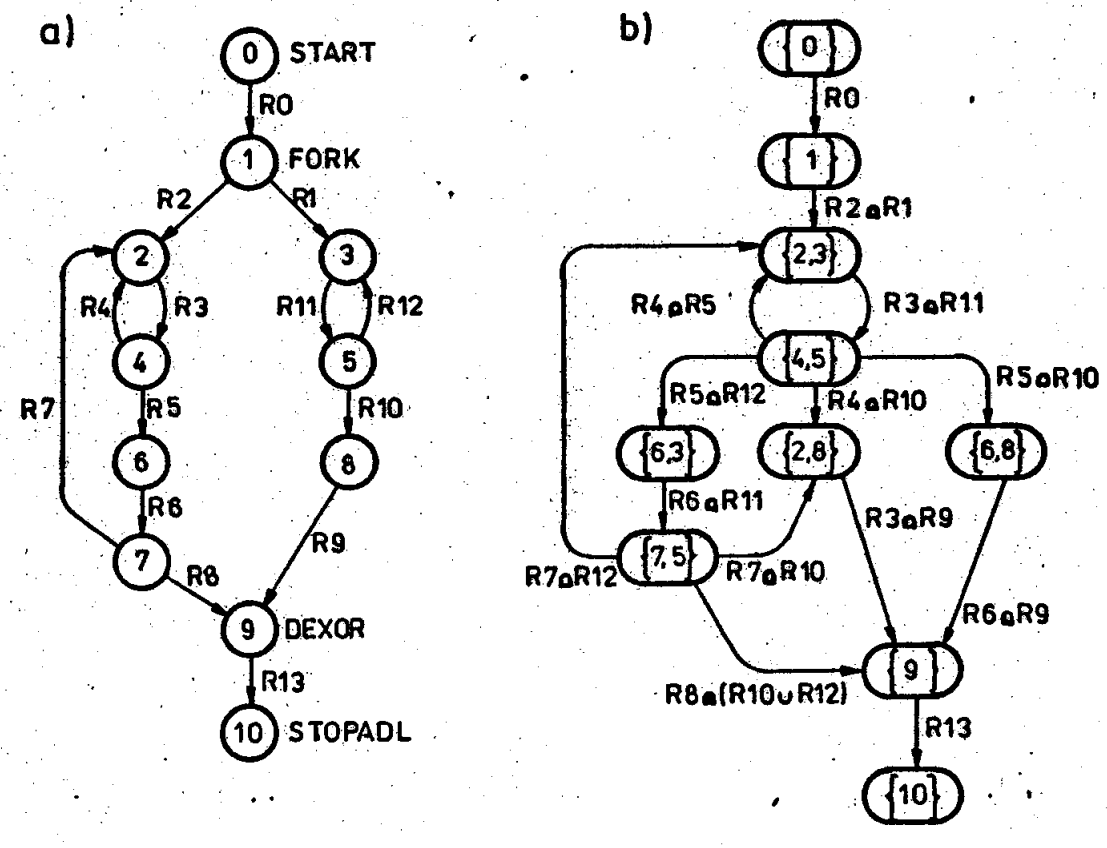


Fig.5.

As an example of parallel program validation we will present validation of an automaton, which has to delay the input on the time T.

Example 2

Problem formulation

A digital device OPTIMP is to be designed for the delay of input signal A by the time T. The device has information input A , information output B , and control input T. A and B are logic signals (logical variables) and T is a parallel variable with value of time T, expressed in the number of main clock CLOCK pulses. In case 1, when delay time is shorter than signal A (the number of pulses when A = 1), the time diagram is presented on Fig.6a.

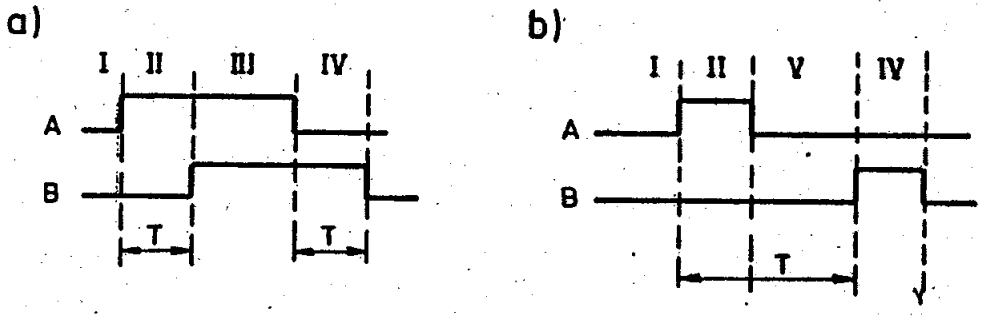


Fig.6.

The diagram on Fig.6b corresponds to case 2 in which the delay time is longer than signal A = 1. It is assumed that the following conditions are fulfilled:

(i) A = 0 in the initial moment ,

(ii) time lag between two signals A = 1 is longer then time T.

According to the above formulation the user writes the program in ADL.

For logical variables (and single bits or Boolean functions playing the role of predicates) we shall adopt the following notation
$(A = 1)$ and $\neg(A = 0)$ is replaced with A, and
$(A = 0)$ and $\neg(A = 1)$ is replaced with $\neg A$.

L and LT are parallel variables.

The translation creates the internal machine representation- the relational structure corresponding to the program-graph of automaton's behavior (Fig.7a). Statements A1 and A2 correspond to the state of control automaton, in which signal A is initially equal to zero (phase I of Fig.6). When A changes to 1 the branching is executed by FORK statement. In one branch ones are added to counter L while $A = 1$ and $LT \neq 0$ (statements A6,A7). In the second branch ones are substracted from the counter LT while $LT \neq 0$ (phase II - statements A4 and A5).

Let us consider case 1 from Fig.6a. If $A = 1$ when LT is set at 0, control goes on to statement A12. Next the device waits while $A = 1$ and gives 1 on output B (statements A9 and A12 - phase III). When A changes to 0, the device counts time of delay stored in counter L, all the time giving 1 on output B (statements A10, A11 - phase IV). After counting the time, device goes to the phase I when L is set at 0 (jump to statement A1).

In case 2 from Fig.6 b the device executes statements A6, A7 in one branch and A4, A5 in another until A will change to 0. Counting up in L counter will be interrupted and counting down will continue in LT counter (statements A4, A5 -phase V). At the moment of LT changing to 0 (i.e. time T has been counted) the value of A is checked. It is equal to 0, so the device executes statements A10 and A11 (phase IV) as in the first case.

The program-graph from Fig.7a is next converted by program CERTIF to the form of parallel relational graph-Fig.7b (the

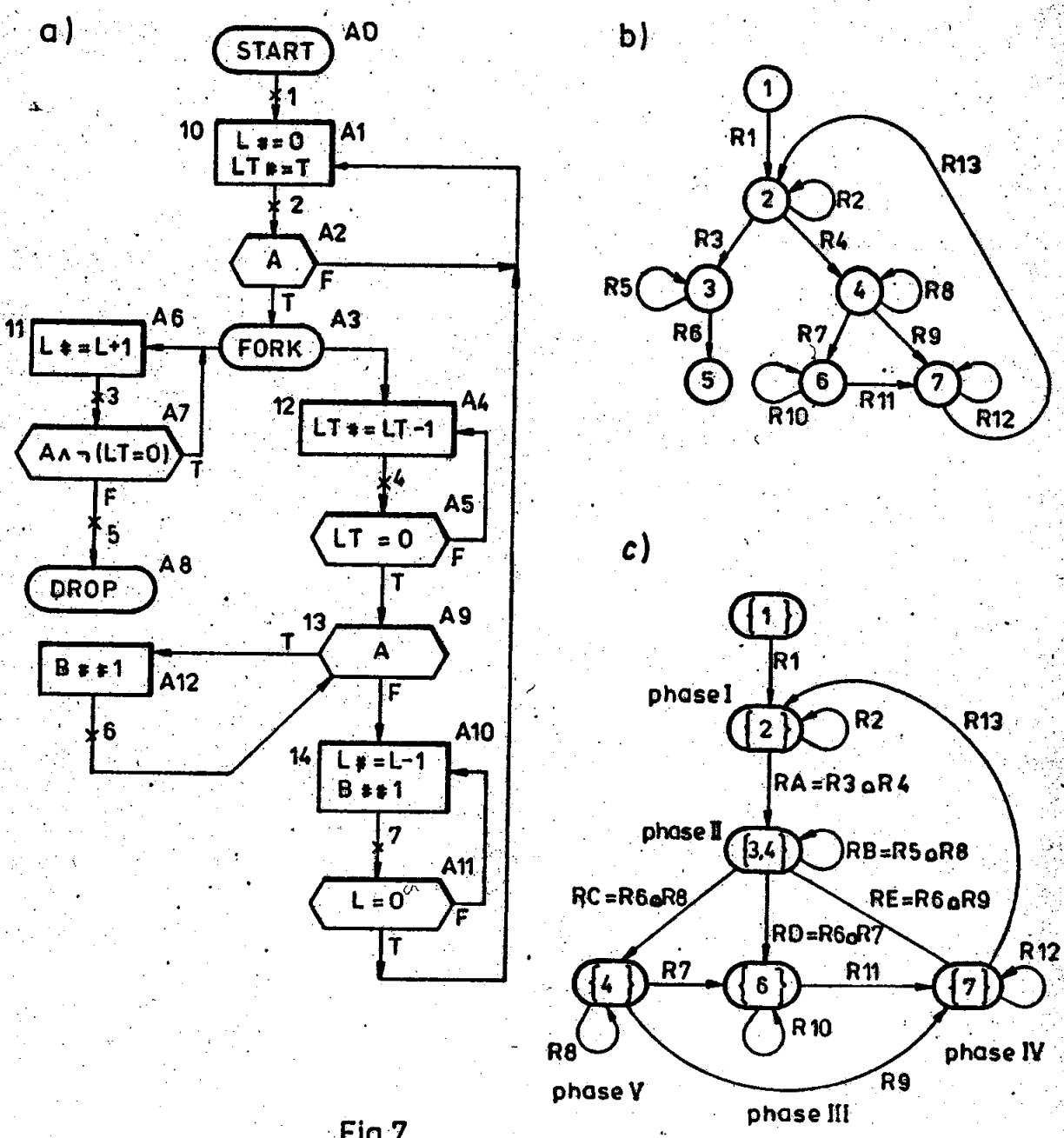

a)
START
A0
1
10
L := 0
LT := T
A1
2
A
A2
F
T
FORK
A3
11
L := L+1
A6
3
A7
A ∧ ¬(LT=0)
T
F
5
DROP
A8
12
LT := LT-1
A4
4
LT = 0
A5
F
T
13
A
A9
T
B := 1
A12
6
F
14
L := L-1
B := 1
A10
7
L = 0
A11
F
T
b)
1
R1
2
R2
R13
R3
R4
R5
3
4
R8
R6
R7
R9
5
6
7
R10
R11
R12
c)
{1}
R1
phase I
{2}
R2
R13
RA=R3∘R4
phase II
{3,4}
RB=R5∘R8
RC=R6∘R8
RE=R6∘R9
RD=R6∘R7
{4}
R7
{6}
R11
{7}
R12
R8
R10
phase IV
phase V
R9
phase III

Fig.7

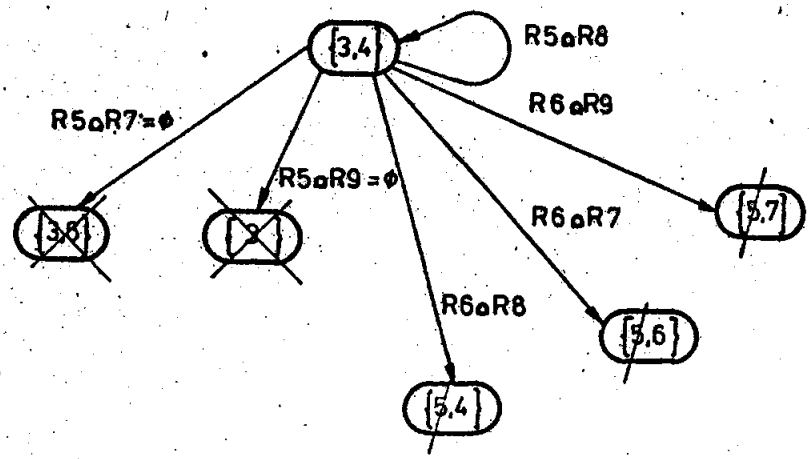

{3,4}
R5∘R8
R6∘R9
R5∘R7=∅
R5∘R9=∅
R6∘R7
R6∘R8
{3,6}
{5,7}
{5,6}
{5,4}

Fig.8

parallelism of this graph results from non-disjoint relations R3 and R4). The meaning of relations R_i in the graph is the following:

R1 = [T | Q1] = [(L, LT, A , B) := (0, T, * , 0)];

R2 = [¬A | Q2]= [¬(A = 1) | (L, LT, A, B): = (0, T, 0, 0)] ;

R3 = [A | Q3] =[(A = 1) | (L, LT, A, B): = (L + 1, LT, 1,0)];

R4 = [A | Q4] = [(A = 1) | (L, LT, A, B): = (L, LT - 1, 1, 0)];

R5 = [A ∧ ¬ D | Q5] = [(A = 1)∧¬(LT = 0) |
(L, LT, A, B):= (L + 1, LT, 1, 0)] ;

R6 = [¬ (A ∧ ¬ D) | Q6] = [¬ ((A = 1) ∧ ¬ (LT = 0)) | (L,LT,A,B) :=
(L,LT, (COND ((D = 1) *) ((D = 0)0)), 0)] ;

R7 = [A ∧ D | Q7] = [(A = 1) ∧(LT = 0)|
(L, LT, A, B): = (L, LT, 1, 1)] ;

R8 = [¬D | Q8] =[¬(LT = 0) | (L, LT, A, B) = (L,LT-1, * , 0)] ;

R9 = [¬A ∧ D | Q9] = [¬(A = 1) ∧ (LT = 0) |
(L,LT,A,B) : = (L - 1, LT, 0, 1)] ;

R10 = [A | Q10] = [(A = 1) | (L, LT, A, B): =(L, LT, 1, 1)] ;

R11 = [¬A | Q11] = [¬(A = 1) | (L, LT, A, B) := (L-1, LT, 0, 1)] ;

R12 = [¬ C | Q12] = [¬(L = 0) | (L, LT, A, B):= (L-1, LT, * , 1)] ;

R13 = [C | Q13] = [(L = 0) | (L, LT, A, B):= (0, T, * , 0)] .

A and B are input and output variables for symbolic tracing.

On the basis of parallel relational graph PRG from Fig.7b the sequential graph RG from Fig.7c is obtained. It is formed as follows. In PRG node $2 \in$ BRANCH from 7 subsets of set $\{R2, R3, R4\}$ only subsets $\{R2\}$ and $\{R3, R4\}$ give non-empty relations(BRANCH is a set of all nodes which disjoin control. All non-empty subsets of set of outpointing relations from node of BRANCH shall be considered as the set of arguments for operation Ω to form a possible arrow in RG.

Thus, because in PRG R3- successor of node 2 is node 3 and R4 - successor of node 2 is node 4 - we get in RG

(R3 ∩ R4) - successor of node $\{2\}$ equal to $\{3,4\}$. The R2- successor of node $\{2\}$ is $\{2\}$.

Figure 8 shows the creation of successors of node $\{3,4\}$. For instance R5- successor of node 3 is node 3 and R9-successor of node 4 is node 7. Thus RG the (R5 ∩ R9) - successor of node $\{3,4\}$ is node $\{3,7\}$. After calculating the value of relation R5 ∩ R9:

$R5 \cap R9 = [(A \wedge \neg D) \wedge (\neg A \wedge D) \mid Q5 \cap Q9 =$

$[F \mid Q5 \cap Q9] =$ NIL, thus the node $\{3,7\}$ does not remain in RG. Similarly

$R5 \cap R7 = [(A \wedge \neg D) \wedge (A \wedge D) \mid Q5 \cap Q7] =$ NIL .

R6-successor of node 3 is node 5 ; R8-successor of node 4 is node 4. Taking into account that 5 ~~∈ corresponds~~ to DROP , we get :

$S(\{3,4\}, R6 \cap R8) = \{4\}$.

All other nodes and arrows of RG can be found in a similar way. The meaning of the relations in RG from Fig.7 is the following.

$RA = R3 \cap R4 = [A \wedge A \mid Q3 \cap Q4] = [(A = 1) \mid (L, LT, \underline{A}, \underline{B}) := (L + 1, LT - 1, 1, 0)]$;

$RB = R5 \cap R8 = [(A \wedge \neg D) \wedge \neg D \mid Q5 \cap Q8] = [A \wedge \neg D \mid Q5 \cap Q8] =$

$= [(A = 1) \wedge \neg(LT = 0) \mid (L, LT, \underline{A}, \underline{B}) := (L + 1, LT - 1, 1, 0)]$;

$RC = R6 \cap R8 = [\neg(A \wedge \neg D) \wedge \neg D \mid Q6 \cap Q8] =$

$= [\neg A \wedge \neg D \mid (L, LT, \underline{A}, \underline{B}) := (L, LT - 1, 0, 0)]$;

$RD = [R6 \cap R7 = [\neg(A \wedge \neg D) \wedge (A \wedge D) \mid Q6 \cap Q7] =$

$= [A \wedge D \mid (L, LT, \underline{A}, \underline{B}) := (L, LT, 1, 1)]$;

$RE = R6 \cap R9 = [\neg(A \wedge \neg D) \wedge (\neg A \wedge D) \mid Q6 \cap Q9] =$

$= [\neg A \wedge D \mid (L, LT, \underline{A}, \underline{B}) := (L-1, LT, 0, 1)]$.

From RG graph a set of equations is compiled, which, after reduction, gives:

$S1 = R1\ S2$; (iii)

$S2 = R2^{*} RA\ RB^{*} [(RC\ R8^{*}\ (R7\ R10^{*}\ R11 \cup R9)) \cup$

$$RD\ R10^{*}\ R11 \cup RE\ ]\ R12^{*}\ R13\ S2 \cup \{R2^{\infty} \cup R2^{*}\ RA\ RB^{\infty} \cup$$
$$R2^{*}\ RA\ RB^{*}\ RC\ R8^{\infty} \cup R2^{*} RA\ RB^{*} (RC\ R8^{*}\ R7 \cup RD)\ R10^{\infty} \cup$$
$$R2^{*}\ RA\ RB^{*}\ [(RC\ R8^{*}\ (R7\ R10^{*}\ R11 \cup R9)) \cup RD\ R10^{*}\ R11 \cup RE\ ]\ R12^{\infty}\}.$$

From the non-halting part of the equation (in $\{\ \}$ parentheses) halting conditions are generated, e.g.:

$$RB^{K \neq \infty} \Rightarrow (\exists K)\ [(LT \leftarrow LT - 1)^{K} = 0 \wedge LT \in NAT \Rightarrow LT_{0} \geqslant 0 \Rightarrow T - 1 \geqslant 0$$
$$\Rightarrow T \geqslant 1$$

The halting part of the equation corresponds to the symbolic state of computation in node S2:

$$S2 = R2^{*}\ RA\ RB^{*}\ RD\ R10^{*}\ R11\ R12^{*}\ R13\ S2 \cup$$
$$R2^{*}\ RA\ RB^{*}\ RC\ R8^{*}\ R9\ R12^{*}\ R13\ S2 \cup$$
$$R2^{*}\ RA\ RB^{*}\ RE\ R12^{*}\ R13\ S2 \cup$$
$$R2^{*}\ RA\ RB^{*}\ RC\ R8^{*}\ R7\ R10^{*}\ R11\ R12^{*}\ R13\ S2 = C1 \cup C2 \cup C3 \cup C4\ . \quad (iv)$$

After evaluating the above relations according to the presented methods, variants C1, C2, C3 and C4 of the symbolic state S2 are created.

$$C1: (L, LT, \underline{A}, \underline{B}) = (0, T, 0^{m}, 1^{T+r+1}, 0, {}^{*}T,$$
$$0^{m+T}, 1^{T+r+1}, 0)\ .$$

The symbolic traces of input variable A and output variable B present a certain input/output time relation ; i.e. on the input sequence $\neg A^{m}, A^{T+r+1}, \neg A, {}^{*}A^{T}$ the automaton answers with the sequence $\neg B^{m+T}, B^{T+r+1}, \neg B$.

The above description of input/output relation corresponds to case 1. Phase I has m pulses, phase II - T pulses, phase III r + 1 pulses, and phase IV - T pulses of CLOCK. (r is some arbitrary number). In case 1 therefore OPIMP automaton operates correctly.

In the same manner, computer evaluates next variants:

$$C2: (L, LT, \underline{A}, \underline{B}) = (0, T, \neg 0^{m}, 1^{n+1}, 0, {}^{*}T - (n+1), 0, {}^{*}T+1$$
$$0^{m+n+1+t+1}, 1^{1+n}, 0)\ ;$$

C3: $(L, LT, \underline{A}, \underline{B}) = (0, T, 0^m , 1^T , 0 , *^T ,$
$0^{m+T} , 1^T , 0) .$

C2 corresponds to case 2 with phase I of m pulses, phase II of n + 1 pulses, phase V of T - (n + 1) pulses and phase IV of n + 1 pulses.

C3 corresponds to case 3, in which length of signal A =(A=1) is equal to T. Phase I has m pulses, phases II and IV - T pulses, phases III and V are absent.

C4: $(L, LT, \underline{A}, \underline{B}) = (0, T, 0^m , 1^{1+n} , 0 , *^u , 1^{v+1} , 0 , *^{p+1} ,$
$0^{m+1+n+u} , 1^{1+v+1+p}).$

This state is next checked against the assumption (ii), which is formally described as:

IMPOSSIBLE-SUBSTRING-OF-VARIABLE-TRACE $(1^{!X} , 0^{!Y} , 1^{?Z} \wedge {}^{!Y} \leqslant T, A)$

which means that the time lag between signals A = 1 is always longer than time T. The goal $(u + 1 \leqslant T)$ is then formulated (the longest possible signal A = 0 can have u + 1 pulses in C4), which matches the fact $(1 + n + u = T)$ resulting from the invariant. If condition (ii) is fulfilled the symbolic state C4 is impossible as having the forbidden trace of the input variable. Automaton operates therefore correctly in cases 1, 2 and 3 if conditions (i) and (ii) are fulfilled, but it does not operate correctly if the same conditions are not fulfilled.

Similarly , other properties of parallel programs of control automata operating in real-time can be proved, and time conflicts between variables can be discovered. Some other applications of CERTIF can be found in [Perk 79 b] and [Stra 77] .

4. THE IMPLEMENTATION

The methods described above form a basis for a system of programs, which is currently being implemented at the Institute of Automatic Control, Warsaw Technical University. After finishing,

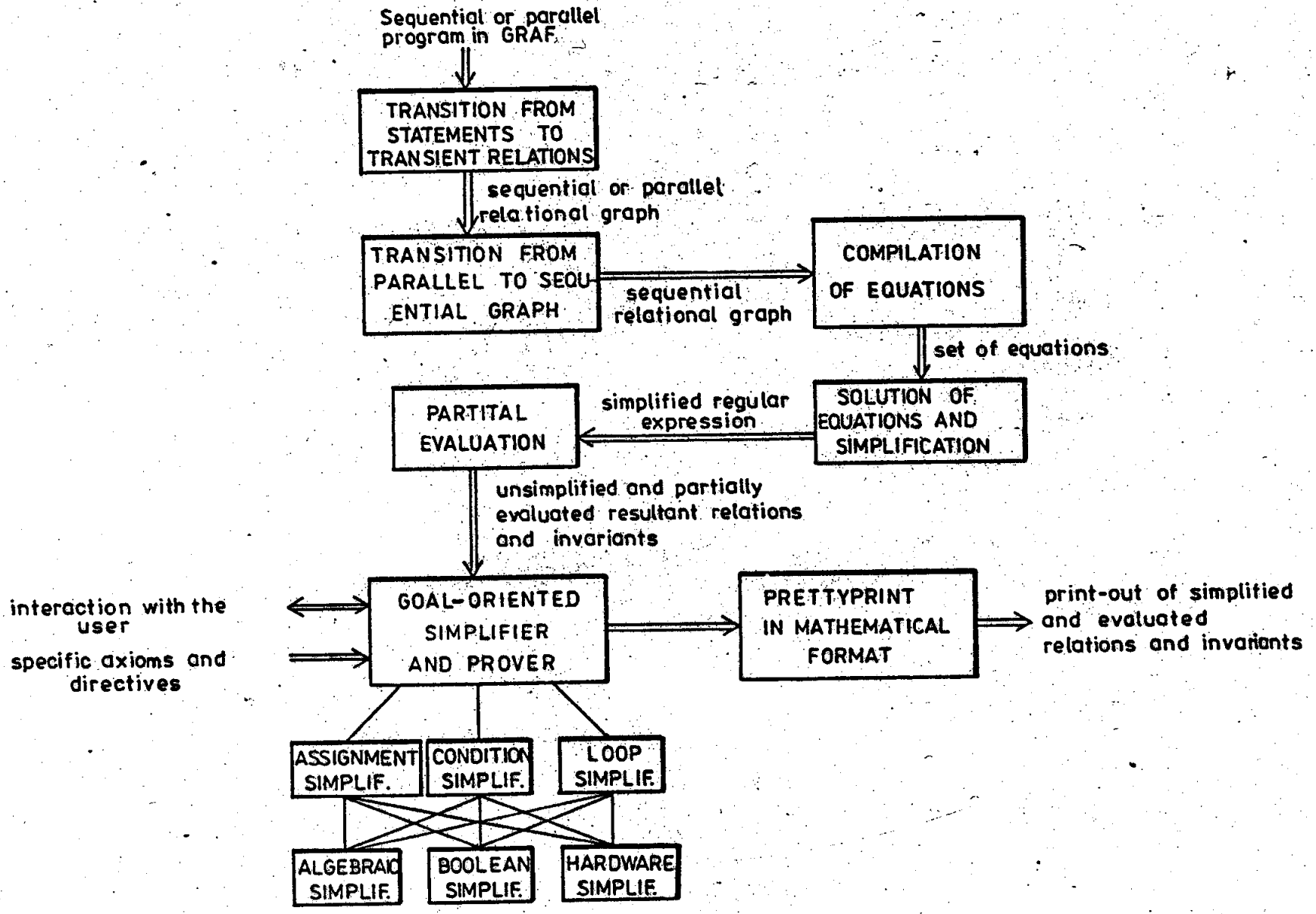

Sequential or parallel program in GRAF
TRANSITION FROM STATEMENTS TO TRANSIENT RELATIONS
sequential or parallel relational graph
TRANSITION FROM PARALLEL TO SEQUENTIAL GRAPH
sequential relational graph
COMPILATION OF EQUATIONS
set of equations
SOLUTION OF EQUATIONS AND SIMPLIFICATION
simplified regular expression
PARTITAL EVALUATION
unsimplified and partially evaluated resultant relations and invariants
interaction with the user
specific axioms and directives
GOAL-ORIENTED SIMPLIFIER AND PROVER
PRETTYPRINT IN MATHEMATICAL FORMAT
print-out of simplified and evaluated relations and invariants
ASSIGNMENT SIMPLIF.
CONDITION SIMPLIF.
LOOP SIMPLIF.
ALGEBRAIC SIMPLIF.
BOOLEAN SIMPLIF.
HARDWARE SIMPLIF.

Fig. 9

the package will be included to the system DIADES [Perk 79a] . The present version (December 1978) of the system is presented on Fig.9. The path from input data (description in GRAF) to non-simplified relations is already in operation, as well as programs for algebraic and Boolean simplification and prettyprint in clear mathematical multiline infix notation. The previous version of the system made use of the resolution-oriented theorem prover [Boru 77] but following that experience, we now prefer to apply the problem-solving language to implement the goal-oriented simplifier and prover. The system is written in LISP 4.1 and PLANNER41 (our improved version of Microplanner for CDC CYBER 73). The simplification program is a mixture of LISP symbolic simplifier (as those used in algebraic programming)and the goal- oriented nondeterministic rule-based simplifier in PLANNER41. The latter is called when some additional information, generated by CERTIF (e.g invariants or symbolic traces), or entered by the designer (e.g. assertions or rules) is required to apply a certain simplification rule. The general form of the rules is: (GOAL ⟨goal⟩ IF ⟨predicate⟩ THEN ⟨action⟩).

5. APPLICATIONS OF SYMBOLIC ANALYSIS IN OPTIMIZATION OF DIGITAL DEVICES

In DIADES, the program-graphs in language GRAF are subject to several transformations, in order to obtain an equivalent variant of program with quasiminimal value of cost function. The superior program generating equivalent variants and selecting the quassiminimal ones is based on the heuristic search in the space of equivalent programs and space of equivalent abstract structures [Perk 79c]. Some of the program-graphs are implemented, which means that the abstract structures (i.e. structures from "abstract blocks" like counters, registers, etc. with non-specified internal structures)

corresponding to the program are generated. Each digital device is described as a hierarchy of elementary systems. Each elementary system can have two interacting units: control unit and operations unit. Control unit controls the operations unit with control signals, according to the program and the predicate signals from the operations unit [Perk 79a] .

The data obtained in the course of the symbolic analysis can often be used in the automatic optimization of the control automaton program, and in consequence - in the optimization of the control unit and/or the operations unit in the corresponding hardware implementation.

Apart from the data presented above, symbolic transient relations computed by similar methods, can be also of value for optimization, especially the relations of segments of the relational graph with certain input and output states.

The following cases are most frequent in the optimization:

1) the state of one condition implies states of other conditions - the result is the simplification of control unit (less input columns in corresponding automaton table - see example 1 in [Glus 70]),

2) the state of one condition remains the same throughout the program - the result being the removal of corresponding program segment (example 1 in [Glus 70]),

3) a condition is equivalent with other condition or some function of conditions- the result is then to select the condition or function of a smaller cost function value,

4) the condition is equivalent with the bit of variable or function of other conditions and variable bits - the result as in 3),

5) the register-transfer statement is unnecessary, because the variable has the same value at the input as it would after statement execution- the result is the removal of this statement,

different one of less value of cost function - this results in cheaper units,

7) generalization of 5) and 6) - the statements sequence can be replaced with another statements sequence with the less value of cost function (example 2 in [Glus 70])*)

These cases are written as a set of rules in LISP, PLANNER41 and modified GRASPE; they constitute a part of program HEOPS for optimizing of control automata programs.

The automatic design at the "register-transfer level of design" is performed by the collaboration of programs HEOPS, OPTY, IMPLEM and CERTIF. HEOPS functions as the supervisor selecting programs and rules for transformations and implementations, OPTY optimizes program-graphs selecting transforming operators. One of them is TWALKER, which looks for similar substructures in the graph of program, organizes program loops in it and introduces the subscripted variables. IMPLEM finds the abstract digital structures corresponding to the programs.

Example 3

Continuing example 1 we will consider the application of the symbolic analysis to the control automaton program's optimization.

When the designer agrees with the resultant relation, invariants and other results of analysis of a certain program, he calls program IMPLEM to generate an operations unit corresponding to that program. The operations unit SAO corresponding to PO program is shown on Fig.10 (for simplicity's sake clock units and connections will not be included).

---

*) The approach to program-optimization based on invariants, similar to some extent to ours, have been also proposed recently by S.Katz [Katz 78].

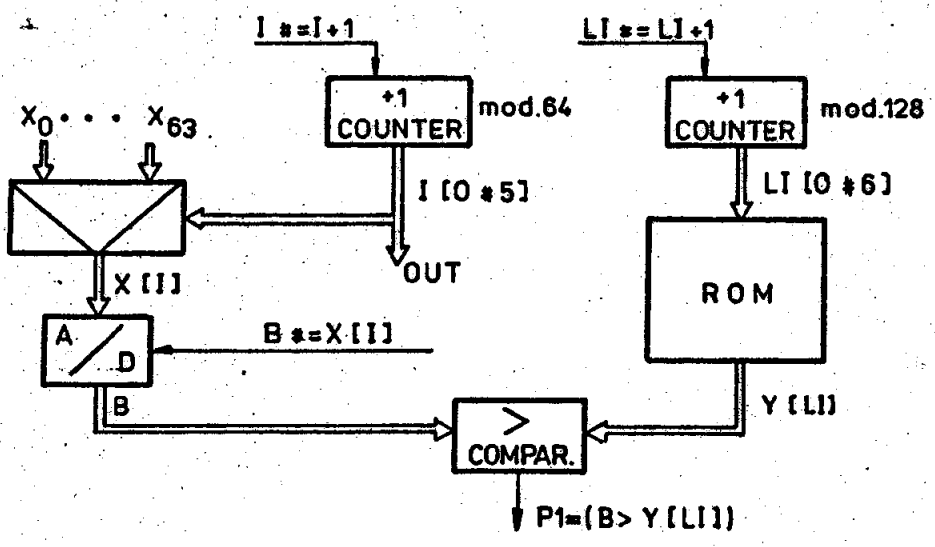


Fig.10

Now control is given to HEOPS program, which tends to find the next, behaviorally equivalent variants of program , and the corresponding operations and control units. In the course of it work HEOPS attempts to find some of the prespecified dependences of the variables. It makes use of the symbolic states of computation evaluated by CERTIF.

From $S1_M$ it gets: I = M and LI = 2 x M.
Then for each M : LI = 2 x I in $S1_M$. Analogically, from $S4_M$ it obtains LI = 2 x I + 1 . In the same way HEOPS proves, that in each Si holds:

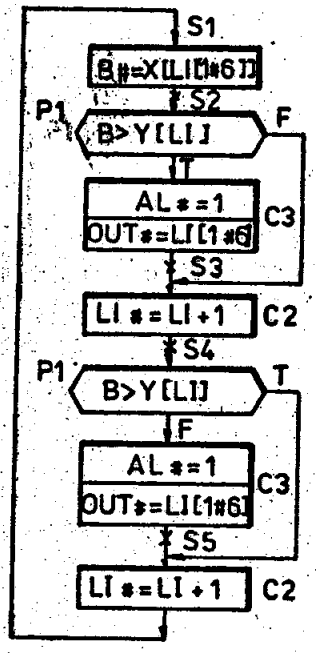
$$LI = 2 \times I \quad \text{or} \quad LI = 2 \times I + 1 \ .$$

Such conclusion leads the program to applying a rule

$$\text{IF } (\exists\, ?K)(\forall Si)\big[\ (2^{?K} \times ?A + ?Y = ?B[0 \# BMAX]\ ) \wedge$$
$$(\ ?Y < 2^{?K}\ )\big] \text{ THEN } \ ?A \longrightarrow ?B[?K \# BMAX]$$

which results in substituting I [0 * 5] with LI [1 # 6] in **P0**. This gives program-graph **P1** from Fig.11, which need not be verified. Next IMPLEM finds abstract structure **SA1** of the operations

unit, corresponding to **P1** (Fig.12). The operations unit is then reduced by one counter.



P1

Fig.11

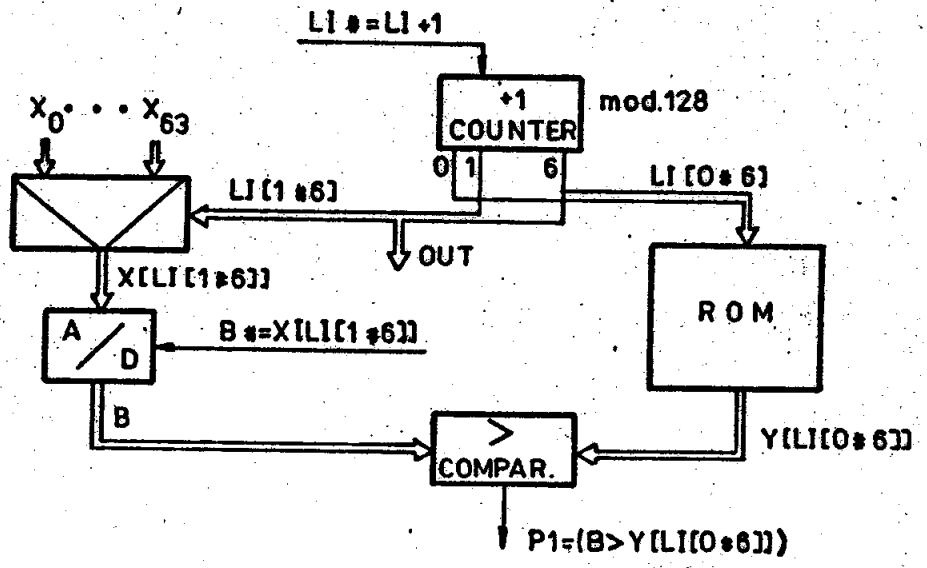


Fig.12

When HECPS sees no more dependences between variables, it gives control to OPTY program, which in turn attempts to reduce the control unit corresponding to program-graph **P1**. Let us assume, that the operator TWALKER which had been called by OPTY has identified two similar substructures in **P1** (the present version of TWALKER is still too weak to notice this type of similarity- see [Perk 79 c]) and next has transformed this program-graph to **P2** (Fig.13), and next to **P3** (Fig.14). C is a control variable i.e. logical,stored variable, used to select the statements of the program-graph.

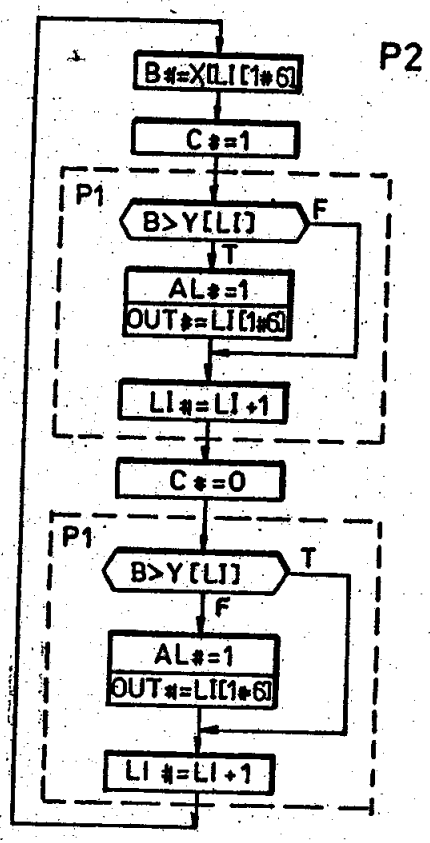


Fig.13.

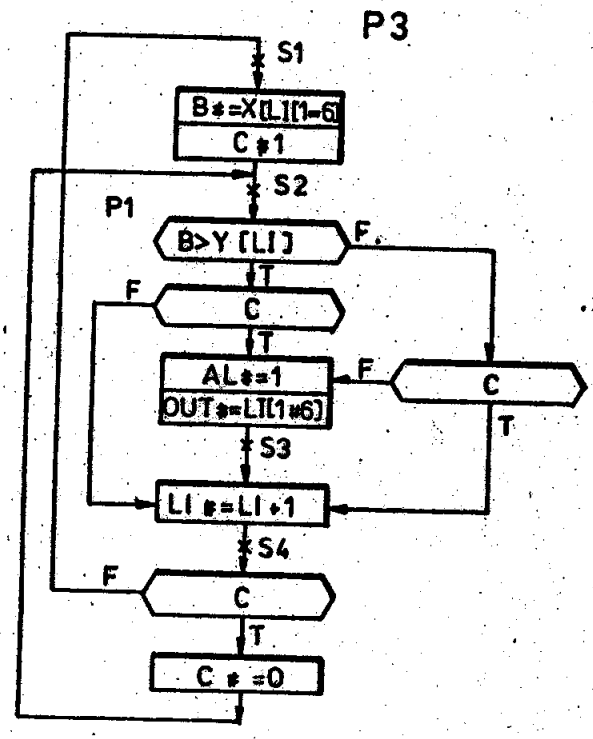
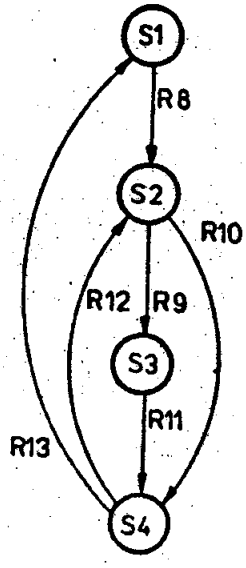


Fig.14

Control is given once more to CERTIF. In order to achieve further reduction of P3 CERTIF makes the corresponding relational graph (Fig.15).



RG4

Fig.15.

Now CERTIF simplifies the condition in R9:

$$P1 \wedge C \vee \neg P1 \wedge \neg C \longrightarrow P1 \equiv C$$

and in R10

$$P1 \wedge \neg C \vee \neg P1 \wedge C \longrightarrow P1 \oplus C .$$

Taking into consideration that $P1 \oplus C \Longleftrightarrow \neg(P1 \equiv C)$, relational graph RG4 corresponding to program-graph P4 from Fig.16 in created.

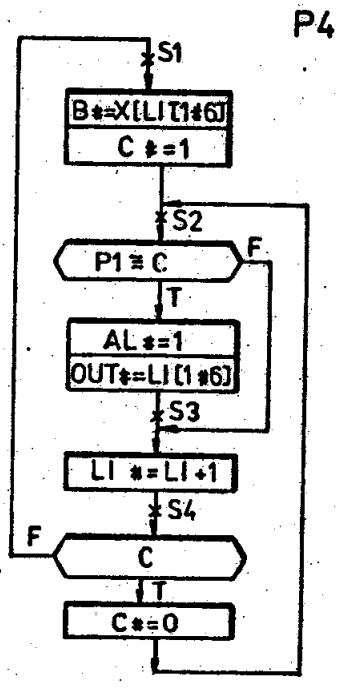


Fig.16.

In order to achieve further reduction of P4 program CERTIF returns control to HEOPS, which investigates the relations among the states of the variables: C , LI and P1 = (B > Y [LI]). To begin with it tries to reduce the number of stored variables; then it tries to simplify the statements. To remove the statements (C #= 1) and (C #= 0) from P4 it must be checked whether the variable C cannot be replaced with a function of LI and P1. CERTIF has to investigate relations between the variables in nodes

S2 and S4, that is prior to the moments in which the state of variable C will decide on the further behavior of program P4. On the basis of relational-graph RG4 CERTIF evaluates resultant relation, invariants and symbolic states similarly as in Example 1. It proves that in state S4 the invariant is:

$$[\,(LI = 2 \times M + 1) \wedge (C = 1) \;\vee\; (LI = 2 \times M + 2) \wedge (C = 0)\,] \wedge \ldots\ldots$$

Then in S4 node condition C is equivalent to condition LI[0 * 0] and directly after that node checking the value of predicate C can be replaced with checking the value of predicate (bit) (LI[0 * 0]) by applying the rule:

IF (∃ control variable ?X) (∃ stored variable ?Y) (∃ ?K ∈ {0,1,...,dim (Y)})[?X ≡ ?Y [?K * ?K] in S]

THEN ?X ⟶ ?Y[?K * ?K] just before S (R7)

The condition following node S4 is then replaced with LI [0 * 0]. Similarly HEOPS program proves that in S2 node condition C is equivalent to (¬ (LI [0 * 0])) , and in condition following node S2 C is replaced with (¬ (LI [0 * 0])). Statements (C *= 0) and (C *= 1) are removed. The resultant condition (P1 ≡ ¬ C) is next replaced with a more simple one (P1 ⊕ C). As a result the program-graph P5 from Fig.17 is obtained.

P5

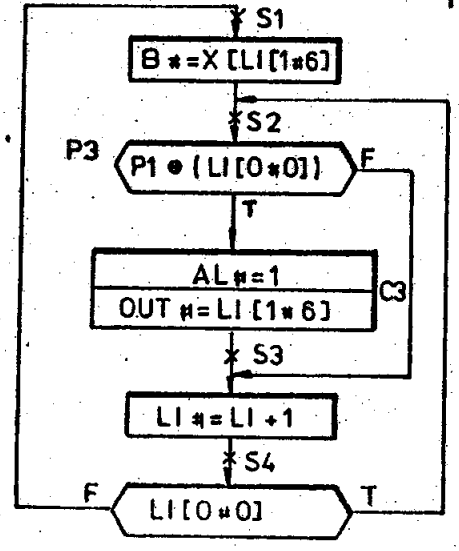


Fig.17.

All programs P0,...,P5 are of course equivalent, because it can be proved that all the transformations applied retain the equivalence of the programs (according to EC1, EC2 and EC4 criteria). There exist also other possibilities of optimization of the above program [Perk 78].

6. CONCLUSION

This paper demonstrates the possibility of practical application of a certain extension of the MB-theory to the symbolic analysis and optimization of sequential and parallel synchronous programs. The methods were succesfully run at several simple examples (including some from [Wegb 74], [Ders 77], [Mazu 74], [Glus 70], [Katz 78]), which required single minutes of CYBER , while the human is much slower at solving such problems and often makes errors. He has of course other abilities; interaction is therefore useful. The practical application of pure evaluation of resultant relations however is limited to small schemata, because with the present strength of the simplifier the resultant data can be unreadable also for certain small programs. It must be pointed out, that even in such case the user can notice some consequences of his program, he could not predict himself, such as the one from Example 2, which specifies that the time lag between two signals A = 1 must be longer than T clock pulses, to ensure proper operation of the circuit. If the user had not introduced condition (ii) he would have been able, in the printed part C4 of the state S2 - to notice its undesirable consequences (unpredicted output sequence). In the cases where the generation of relations, states and invariants does not give readable results, the designer still has the possibility of generating first term expansions of the relations, which can be of much help in inductive inferring of relations or

invariants and in debugging the program.

An interesting, though quite natural; property of the approach is that programs without go-to's (in a sense - structural) have simpler regular expressions,and resultant relations which are more readable and easier to handle.

The presented methods can be also applied as a partial tool in inductive invariants generation and in some problems of symbolic simulation [Lewi 79] . The equivalence of programs can be reduced to the equivalence of resultant relations (or equality in special case). Additional counters in the program-graph are used to estimate program complexity, and in real-time analysis to simulate time. The proposed methods cover also some techniques for optimization given in [Glus 70] and in the papers of other specialists, in the so-called theory of logical schemes of algorithms (LSA) [Ersh 73] . These problems are studied in more details in [Perk 79 b] , and also by some other members of our group.

REFERENCES

[Birm 74] Birman A.: "On Proving Correctness of Microprograms". IBM J.Res.Develop.Vol.18, May 1974. pp.250-266.

[Blik 72] Blikle A., Mazurkiewicz A.: "An Algebraic Approach to the Theory of Programs, Algorithms,Languages and Recursiveness". Proc.of an.Intern.Symp.and Summer School on Math.Foundations of Computer Science. Warsaw-Jabłonna, August 21-27, 1972.

[Blik 73] Blikle A.: "An Algebraic Approach to Programs and their Computations". Proc of II Symp.and Summer School on MF of CS. High Tatras, Sept. 3-8, 1973.

[Blik 76] Blikle,A., Budkowski,S.: "An Algebraic Program Verification Method Applied to Microprograms". Res.Rep. CS-76-31 Univ.of Waterloo, June 1976.

[Perk 76] Perkowski,M., Jankowski, K.: "Validation and Optimization of Control Automata Programs in the System for Automatic Design". Proc.of the Intern.Symp.Fault Diagnosis of Digital Networks and Fault-Tolerant Computing, Wisła, Poland, 25-27 May 1976, pp.104-112, Katowice, 1976.

[Perk 77] Perkowski, M.: "A Method of Validation of Parallel Programs in the System for Automatic Design of Block-Oriented Digital Systems". Proc. 2nd IFAC Symp.Discrete Systems. Dresden, GDR, 14-19 March 1977, Vol.2, pp.71-88, 1977.

[Perk 78] Perkowski, M: "Some Selected Problems of Automatic Design of MOS LSI Digital Systems". Inst.Autom.Control, Warsaw Techn.Univ., Warsaw, 1978 (in Polish).

[Perk 79a] Perkowski,M.: "Design Automation of Digital Systems as a New Domain for AI Research". Submitted to IJCAI 6. 1979. Also Technical Report.Inst.Autom.Control,Warsaw Techn.Univ.,Warsaw, 1979.

[Perk 79b] Perkowski,M.: "The Programming System for Symbolic Analysis, Optimization, and other Problems Related to the Semantics of Control Programs". Technical Report,Inst. Autom.Control,Warsaw, Techn.Univ.,Warsaw, 1979.

[Perk 79c] Perkowski,M.: "Digital Design by Problem-Solving Transformations" Submitted to IJCAI 6. 1979. Also Technical Report, Institute of Automatic Control, Warsaw Technical Univ. 1979.

[Stra 77] Strawiński,T.:"Mazurkiewicz-Blikle Method in Automatic Optimization of Programs".Inst.Autom.Control, Warsaw Techn.Univ., Warsaw 1977 (in Polish).

[Wagn 77] Wagner, T.J.: "Hardware Verification". Stanford Artif. Intell.Lab., Memo AIM -304, CS.Dept.Report No.STAN-CS-77-632, September 1977.

[Wegb 74] Wegbreit,B: "The Synthesis of Loop Predicates". CACM, Vol.17, No.2, pp.102-112, 1974.