INTERNATIONAL COMPUTER SCIENCE INSTITUTE

1947 Center St. * Suite 600 * Berkeley, California 94704-1198 * (510) 643-9153 * FAX (510) 643-7684

Positional Logic Algebra
- PLA -

A Fascinating Alternative Approach

Christian - M. H a m a n n ⋆
Lev C h t c h e r b a n s k i ⋄

TR-97-039

September 1997

Abstract

The Russian researcher, M. Telpiz, presented 1985 in Russia a totally new approach to logic algebra (and L. Chtcherbanski, a friend of his, brought the ideas to Berlin/Germany in 1995). PLA may be an elegant and better representation for some problem domains then the Boolean Algebra. Highlights of PLA are:

- Only one simple algorithm holds for all calculations
- Invert operators build invert functions
- Operators are directly applicable on operators and therefore
- Compilation of multi-layer networks are possible via simple calculations over operators only

PLA has a potential for new applications in logical calculus problems, specially with many variables. Because operators are directly applicable on operators, PLA may be of special interest in research areas of Genetic Algorithms, Evolution-Strategy and Artificial Life.

⋆ Informatik/KI-Labor, TFH-Berlin, Germany hamann@tfh-berlin.de
⋄ Bionik/Evolutions-Strategie, TU-Berlin, Germany lev@tfh-berlin.de

Abstract

The Russian researcher, M. Telpiz, presented 1985 in Russia a totally new approach to logic algebra (and L. Chtcherbanski, a friend of his, brought the ideas to Berlin/Germany in 1995). PLA may be an elegant and better representation for some problem domains then the Boolean Algebra.

Highlights of PLA are:

- Only one simple algorithm holds for all calculations
- Invert operators build invert functions
- Operators are directly applicable on operators and therefore
- Compilation of multi-layer networks are possible via simple calculations over operators only

PLA has a potential for new applications in logical calculus problems, specially with many variables. Because operators are directly applicable on operators, PLA may be of special interest in research areas of Genetic Algorithms, Evolution-Strategy and Artificial Life.

1 Introduction

In solving logic calculations with conventional Boolean Algebra the exponential increase in complexity with the increasing number of variables is a sound problem. The "Positional Logic Algebra" (PLA), first presented by M.Telpiz [Tel-85] in 1985, promised to give tools for braking down this problem.

Each logic algebra deals with the functional dependencies of the values from the logic variables and the values from the logic function. One can say: A function is a transformation from one set into another set. If we have a set of variables with two-values {0,1} and a function which produces a new set with two-values {0,1}, then this function is called a logic function in dual logic or Boolean Logic. The ground-functions of Boolean Algebra are:

X	Y	X AND Y	X OR Y
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

Z	NOT Z
0	1
1	0

Tab.1 Truth-tables of AND, OR and NOT

The truth-tables shown above, are the best-known functions AND and OR, connecting two logic variables X and Y, and the function NOT for negation of a variable Z. A logic function of many variables is defined by a set of terms. Each term is a unique n-ary combination of zeros and ones. Following the convention, the terms are sorted in dual-code sequence. The set of all terms (associated with the function-value) is called the logic function.

With these three logic ground-functions AND, OR and NOT all other functions of the dual logic can be made. But the function NAND (NOT-AND, Sheffer Stroke) alone or the function NOR (NOT-OR, Peirce Function) alone could be used as ground-functions.

Nr.	X=0 0 1 1 Y=0 1 0 1	Boolean Operator	equivalent ground-function	mathematical description	short	colloquial language
0	0 0 0 0	0		Contradiction	FALSE	never
1	0 0 0 1	$X \wedge Y$		Conjunction	AND	and
2	0 0 1 0		$X \wedge \sim Y$	Inhibition2		
3	0 0 1 1	X		Projection1		
4	0 1 0 0		$\sim X \wedge Y$	Inhibition1		
5	0 1 0 1	Y		Projection2		
6	0 1 1 0	$X \neq Y$	$(X \wedge \sim Y) \vee (\sim X \wedge Y)$	Disvalence	XOR	either-or
7	0 1 1 1	$X \vee Y$		Disjunction	OR	or
8	1 0 0 0	$X * Y$	$\sim X \wedge \sim Y$	Peirce-Func.	NOR	neither-nor
9	1 0 0 1	X = Y	$(X \wedge Y) \vee (\sim X \wedge \sim Y)$	Equivalence	EQ	if-and-only-if
10	1 0 1 0	$\sim Y$		Negation2		
11	1 0 1 1	$X \leftarrow Y$	$X \vee \sim Y$	Implication1		then-if
12	1 1 0 0	$\sim X$		Negation1		
13	1 1 0 1	$X \rightarrow Y$	$\sim X \vee Y$	Implication2		if-then
14	1 1 1 0	$X \mid Y$	$\sim X \vee \sim Y$	Sheffer-Func.	NAND	all-not
15	1 1 1 1	1		Tautology	TRUE	ever

Tab.2 All possible Boolean functions for two variables X and Y in systematically order

2 Positional Logic Algebra

The following example shows, how the value of a logical function P(X) with the variables X1 ... X4 is calculated in the Positional Logic Algebra (PLA) with the Positional Operator **p**.

In the line of these 4 variables X1 ... X4 we mark 4 + 1 positions: "In-front-of", "between 1 ... 3" and "behind". The positions are labeled with symbols ":" and "-". For example:

P(:X1-X2:X3-X4-)
$\wedge \quad \wedge \quad \wedge \quad \wedge \quad \wedge$
$0 \quad 1 \quad 2 \quad 3 \quad 4 \Leftarrow$ position number k

The Positional Operator **p** is the sequence of the labels ":" (=1) and "-" (=0). The bracketed values (1 or 0) make it possible, to apply operators on operators (as shown further below).

p = (: - : - -) = (10100)

2.1 Simple Positional Operators

Inside the operator the label ":" in position k0, k1, k2, ... , ks means: The line with the values of the variables X1 ... Xn contains k0 (= no) "1" or k1 or k2 or ... ks "1". The value of the function P(X1 ... Xn) is equal to the label in the operator **p** (in 0;1 representation) on

the position number which equals the sum of 1 in this line. An example:

Function P(:X1-X2:X3-X4-) with operator p = (: - : - -) = (10100)					
X1	X2	X3	X4	P	Remarks
0	0	0	0	1	No times "1", that's why pos. "0" gives us "1"
0	0	0	1	0	one times "1", that's why pos. "1" gives us "0"
0	0	1	0	0	one times "1", that's why pos. "1" gives us "0"
0	0	1	1	1	two times "1", that's why pos. "2" gives us "1"
0	1	0	0	0	one times "1", that's why pos. "1" gives us "0"
0	1	0	1	1	two times "1", that's why pos. "2" gives us "1"
0	1	1	0	1	two times "1", that's why pos. "2" gives us "1"
0	1	1	1	0	three times "1", that's why pos. "3" gives us "0"
1	0	0	0	0	one times "1", that's why pos. "1" gives us "0"
1	0	0	1	1	two times "1", that's why pos. "2" gives us "1"
1	0	1	0	1	two times "1", that's why pos. "2" gives us "1"
1	0	1	1	0	three times "1", that's why pos. "3" gives us "0"
1	1	0	0	1	two times "1", that's why pos. "2" gives us "1"
1	1	0	1	0	three times "1", that's why pos. "3" gives us "0"
1	1	1	0	0	three times "1", that's why pos. "3" gives us "0"
1	1	1	1	0	four times "1", that's why pos. "4" gives us "0"

Tab.3 Example of how to get the value P out of the operator p

With these "Simple" Positional Operators for n variables one can formulate $2^{(n+1)}$ logic functions. The whole number of dual logic functions with n variables is however $2^{(2^n)}$. This means that with Simple Positional Operators only a subset of dual logic functions can be represented.

This subset contains the well known symmetric logic functions. They are called symmetric, because the value of the logic function holds, independent of the permutation of the input variables.

If we write down all possible Simple Positional Operators for two variables, we find the following functions:

P0(-X1-X2-), P1(-X1-X2:), P2(-X1:X2-), P3(-X1:X2:), ... P7(:X1:X2:)

Each Simple Positional Operator can be described in a binary code. Take all possibilities for two variables and put it in order of dual code. We get 000, 001, 010, 011, 100, 101, 110, 111. Let's name the operators p0 ... p7 the functions P0 ... P7 and let's make Tab.4.

In analyzing Tab.4, it shows that operators and function values of the left side are mirrored to those of the right side of the table and that the inverted operator generate the appropriate inverted function!

Operator:		p0	p1	p2	p3	:	p4	p5	p6	p7
X1	X2	000	001	010	011	:	100	101	110	111
0	0	0	0	0	0	:	1	1	1	1
0	1	0	0	1	1	:	0	0	1	1
1	0	0	0	1	1	:	0	0	1	1
1	1	0	1	0	1	:	0	1	0	1
Operator:		FALSE	AND	XOR	OR	:	NOR	EQ	NAND	TRUE
Function:		P0	P1	P2	P3	:	P4	P5	P6	P7

Tab.4 Symmetric properties of PLA operators for two variables

It is also remarkable, that for calculations of all functions only one simple algorithm is used: First calculate the number of "1" in the term and then choose the symbol {0;1} according to the position in the operator.

If we write down all possible functions Q0 ... Q3 for one variable Z, we get:

Operator:	q0	q1	q2	q3
	00	01	10	11
Z	(-Z-)	(-Z:)	(:Z-)	(:Z:)
0	0	0	1	1
1	0	1	0	1
Operation:	FALSE	Z	$\sim Z$	TRUE
Function:	Q0	Q1	Q2	Q3

Tab.5 All possible PLA functions for one variable

The special functions and its operators are:

Q1(-Z:) q1 = (- :) = (01) "IDENTITY"
Q2(:Z-) q2 = (: -) = (10) "NOT"

Example 1: The Parity Detector for 32 Bit

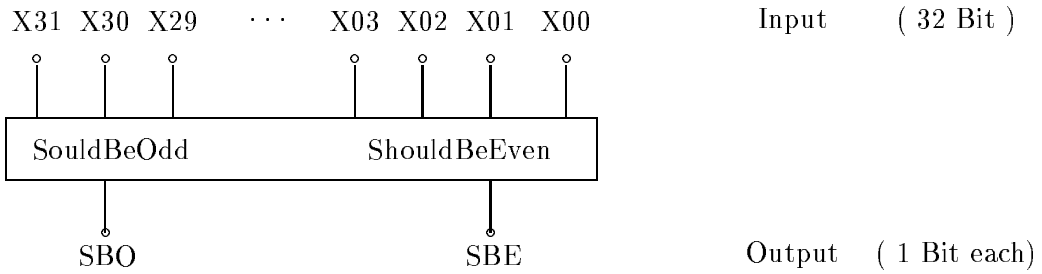

Fig.1 32 Bit Parity Detector

The "inner" PLA function to calculate parity is P:

P(-X31:X30-X29: ... -X03:X02-X01:X00-)

With the IDENTITY and NOT we calculate the output functions:

SBO(-P:) and SBE(:P-)

Example 2: Multi-layer Calculations / Compilations

Positional Logic Algebra allows the application of operators on operators in order to get new operators. In Boolean Algebra this is not possible: A construction like (AND (OR AND)) is in Boolean Algebra meaningless. The following example shows, how the operator **b** (the function B) is calculated by application of operator **c** on the operators **ai**.

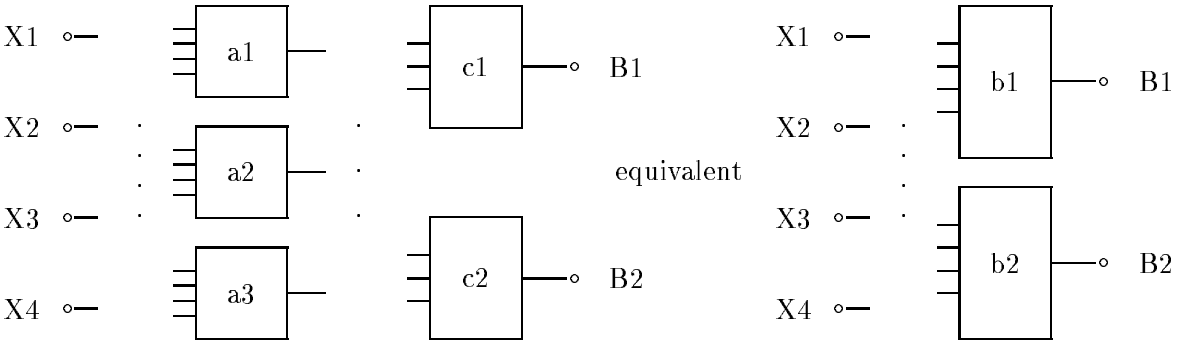

Fig.2 Example of how to compile a multi-layer in a single-layer network
(Dots mean: Connect left-side points with all right-side points)

Look at an example of the functions A1, A2 and A3 with the operators a1, a2 and a3 given:

A1(-X1-X2:X3:X4-) a1 = (00110)
A2(-X1:X2-X3-X4:) a2 = (01001)
A3(:X1-X2-X3:X4:) a3 = (10011)

Lets make the operators c1 and c2

c1 = (0101)
c2 = (1010)

With the functions C applied on the functions A we will get functions B (as shown in Fig.2):

C1(-A1:A2-A3:) = B1(X1 ... X4)
C2(:A1-A2:A3-) = B2(X1 ... X4)

For compiling the network, we search for the operators b1 and b2 and find them by application of operators c1 and c2 on operators a1...a3 in handling these like variables!

k	a1	a2	a3	s	b1	b2
0	0	0	1	1	1	0
1	0	1	0	1	1	0
2	1	0	0	1	1	0
3	1	0	1	2	0	1
4	0	1	1	2	0	1

Tab.6 see text

In Tab.6 **k** is the position number of the operator **ai** (written as columns). **s** is the number of "1s" in the lines of **ai**. These refer to the positions in c1 resp. c2. One get operator **b** equal "1", if there is a colon (resp. 1) in operator c, or "0", if there is an minus (resp. 0) in it. So we find:

B1(:X1:X2:X3-X4-) b1 = (: : : - -) = (11100)
B2(-X1-X2-X3:X4:) b2 = (- - - : :) = (00011)

In calculating the whole truth-table, one can prove the result. One can see:

- With this easy received operator **b** applied on the variable X one can get the function B directly without detour via A and C
- The operators c1 and c2 are inverted; the results of B1 and B2 are inverted also.

X1	X2	X3	X4	A1	A2	A3	B1	B2
0	0	0	0	0	0	1	1	0
0	0	0	1	1	0	0	1	0
0	0	1	0	1	0	0	1	0
0	0	1	1	0	1	0	1	0
0	1	0	0	1	0	0	1	0
0	1	0	1	0	1	0	1	0
0	1	1	0	0	1	0	1	0
0	1	1	1	0	1	1	0	1
1	0	0	0	1	0	0	1	0
1	0	0	1	0	1	0	1	0
1	0	1	0	0	1	0	1	0
1	0	1	1	0	1	1	0	1
1	1	0	0	0	1	0	1	0
1	1	0	1	0	1	1	0	1
1	1	1	0	0	1	1	0	1
1	1	1	1	1	0	1	0	1

Tab.7 Truth-table of the multi-layer network

2.2 Complex Positional Operators

Unsymmetrical logic functions can be traced back to the symmetrical ones (see Boolean table, Tab.2) via transfer functions. For example the two Implications:

IF X THEN Y $\Longleftrightarrow$ (NOT X) OR Y resp. $X \rightarrow Y \Longleftrightarrow \sim X \vee Y$
X THEN IF $Y \Longleftrightarrow X$ OR (NOT Y) resp. $X \leftarrow Y \Longleftrightarrow X \vee \sim Y$

The function OR for two variables was (see Tab.4):

P3(-X1:X2:) p3 = (-::) = (011) "OR"

The functions NOT for one variable was (see Tab.5):

Q2(:X0-) q2 = (:-) = (10) "NOT"

With OR and NOT and the transfer functions we get the two possible Implication functions with "Complex" Positional Operators:

Pi2(-(:X-):Y:) pi2 = (-(:-)::) = (0(10)11) "Implication2"
Pi1(-X:(:Y-):) pi1 = (-:(:-):) = (01(10)1) "Implication1"

With the following truth-table, one can get the proof:

X Y	(:X-)	(:Y-)	(-(:X-):Y:)	(-X:(:Y-):)
0 0	1	1	1	1
0 1	1	0	1	0
1 0	0	1	0	1
1 1	0	0	1	1
X Y	$\sim X$	$\sim Y$	$\sim X \vee Y$ $X \rightarrow Y$	$X \vee \sim Y$ $X \leftarrow Y$

Tab.8 Truth-table to proof the Implications

3 Positional Logic Algebra vs. Boolean Algebra

What is the relation between Positional Logic Algebra and Boolean Algebra? As shown above (Tab.4), the ground-functions AND, OR and NOT and also the functions NOR and NAND are representable with Simple Positional Operators. Tab.9 is summerizing all symmetrical Boolean functions on two variables.

We can say: The Boolean Algebra contents the Positional Logic Algebra. PLA may be an elegant and better representation for some problem domains then the Boolean Algebra.

P0(-X1-X2-)	p0 = (- - -) = (000)	"FALSE"
P1(-X1-X2:)	p1 = (- - :) = (001)	"AND"
P2(-X1:X2-)	p2 = (- : -) = (010)	"XOR"
P3(-X1:X2:)	p3 = (- : :) = (011)	"OR"
P4(:X1-X2-)	p4 = (: - -) = (100)	"NOR"
P5(:X1-X2:)	p5 = (: - :) = (101)	"EQ"
P6(:X1:X2-)	p6 = (: : -) = (110)	"NAND"
P7(:X1:X2:)	p7 = (: : :) = (111)	"TRUE"

Tab.9 All possible PLA-functions and PLA-operators for two variables

Summarizing the highlights of PLA:

- Only one simple algorithm holds for all calculations
- Invert operators build invert functions
- Operators are directly applicable on operators and therefore
- Compilation of multi-layer networks are possible via simple calculations over operators only

PLA has obviously a potential for new applications in logical calculus problems, specially with many variables.

Because operators are directly applicable on operators, PLA may be of special interest in the research areas of Genetic Algorithms, Evolution-Strategy and Artificial Life.

4 References

[Tel-85] Telpiz, M.
"Representation of functions with a logical algebra"
Cybernetica 4(1985)37-40 (in Russian)

[Ch-96] Chtcherbanski, L. and Hamann, C.-M.
"Positionelle Logische Algebra -
Eine Alternative mit vielversprechenden Moeglichkeiten"
VDI-Brandenburg, Teltower-Innovations-Tage, 9.Nov.1996

5 Appendix

In the appendix there is a copy of E-Mail we received in June 1997 from the Telpiz-group. He is founding a company in Sweden to commercialize his ideas and he gives some aspects of possible applications he has in mind.

START OF E-MAIL QUOTATION:

****

Extended Computer Logics: ECL System AB
Org. Num. 556521-6438
Box 12095
40241 Goteborg
Sweden

Company Profile And Perspectives in April 1997

The Swedish company ECL System AB formed in November 1996 is developing an exciting new technology called digital logics (or positional logics). The technology is based on the published new results in fundamental mathematics but it extends the theory that existed previously and adapts it for the solution of practical problems in various application domains. In contrast to the classical logics and programming systems based on it, in digital logics, knowledge is stored much like the decimal numbers are used in everyday life or as binary numbers are stored in computers. The position-based representation of logical formulas uses a special alphabet with symbols (productions) corresponding to high-level transformations of the logical formulas. The action of these transformations is dependent on their position in the combined operator (clause). It is important that this representation of logical formulas is extremely compact. For example, to express the non-equality of N-bit numbers takes just 1 clause instead of 2^N clauses in the corresponding disjunctive normal form (a conventional way of representing constraints in logics and arithmetics). The digital representation of logical information is usually a conjunction of clauses (a conjunctive form, CF), which can be manipulated upon quite efficiantly by using algebraic operations and well-defined algorithms.

The company has developed a number of algorithms starting with SAT-solver (used to solve the problems expressed in conjunctive normal form, CNF) characterized by extremely small and constant memory requirements and completeness, all the way through to algorithms using progressively larger number of positional logic productions.

The application domain of digital logics is very wide. Significant gains can be achieved in almost all problems known as NP-complete or difficult problems in discrete mathematics. Below is the list of the potential application areas where excellent results can be obtained quite fast.

1. **Knowledge Learning:**
   This is a very direct application of the digital logics technology. Logical knowledge learning is different from the neural networks because it stores the information as logic formulas that can be directly interpreted as expert advices of 'professionals' in the field. This has parallels to constructing a program given the inputs and outputs, discovering a law of nature based on obsevations, discovering what an electronic black box is based on sets of its inputs and outputs, a game AI engine discovering the strategy the player uses, etc. Expert systems can be constructed automatically this way. Usually, the logical information is stored as decision trees or decision lists. Given the highly expressive

power of the digical logics, one can construct (or 'learn') similar but extremely compact representations of the knowledge when the system is presented with examples of correct or desirable course of events.

2. **Electrical engineering:**
   Circuits design, logic synthesis, design verification. The technology can be used for conventional synthesis, but also, an entirely new type of electronic devices can be developed. The characteristic properties of those would include compact size, reversibility, speed, extensibility, testability, and direct combination of logics, arithmetics, and control. One result obtained by our research team is of particular importance to electronics applications: A special reduced representation of the conjunctive form allows for output of solution vectors in linear time, and given an enormous expressive power of positional logics formulas, this form is quite compact, and lends itself for direct hardware implementation.

3. **Integer linear programming, integer-domain equations:**
   Addressing these types of hard (NP-complete) problems by converting them to conjunctive normal form (CNF) leads to exponential texts, and, obviously, exponential solution times. Suppose you want to write down the constraint that the two 5-bit numbers are equal if a certain flag is set: $f \Rightarrow (a == b)$. In conjunctive normal form (CNF) one writes $2^5 = 32$ clauses, while only one positional logics clause is sufficient for 5 bits and any larger number of bits. Moreover, the clauses that constitute the problem can be a conjunction of arithmetic, logical constraints (like a+3b<4, 15c-a>3, d=(a=b), (d OR g)⇒(t1 OR t2)), and special control statements. The mixture of linear programming and logical constraints and control has enormous application potential, and we would like to see concrete customers problems in this field. We are also in the process of writing a general-purpose integer linear programming solver.

4. **Recource allocation, scheduling (including multi-processor scheduling):**
   These areas typically require the problems of graph-coloring type to be solved efficiently. The specialized algorithms are under development that will find the solutions very fast. In particular, complete versions of algorithms make it possible to prove that a given coloring does not exist. By combining the algorithm with conventional techniques, we expect to achieve record performance.

5. **Optimal classification (discretization) problem:**
   Given the set of multi-parameter observations on a system, find optimal classifiers (minimum number of planes) that divide the multi-dimensional parameter space in boxes with one or no observation in each box. This important step is needed to reduce the multi-parameter real-valued information about the complex system for feeding into any type of decision-making system (for example, the above-mentioned a decision-tree system or a neural network).

6. **Theorem proof and first-order logic engines.**

In addition to the development for the mentioned immediate applications of the technology, the company is extending its lead in the area of computational logics by conducting extensive research. The ultimate goal is the construction of the complete set of productions that should achieve linear computation times (provided the problem is written down using these operators) and open the perspective for the computers capable of reasoning. The other future plans include the development of extremely fast algorithms for large number factorization problems.

ECL System AB will now be concluding cooperation agreements with some European high-tech companies (including transportation, communications, and medical companies). The company will also be negotiating a special agreement with IBM Research in US to implement a new technology for digital circuits verification.
...

Discussion Items for the Forthcoming Meetings with Companies in May 1997

1. **Arithmetization of logics:**
   Functions are stored in a positional way resembling numbers in any positional system (decimal, binary...). The 'digits' in the alphabet are called productions, the 'numbers' are called operators. There is a large class of logical and arithmetical functions that can be expressed as one operator. Any operator can be negated in linear time. All logical functions can be expressed as collections (tables) of operators with different logical operations between them. Generalized resolution infers knowledge by transforming collections of operators into a form that excludes conflicts in the full set of symmetrically equivalent failure paths; this form contains all solutions to the problem and can generate them in linear time. Obviously, one operator generates solutions in linear time.

2. **Important recent engineering results:**
   Any linear system in GF(2) (a conjunction of parity functions) is expressed as one operator; moreover, any negation, disjunction, or conjunction of linear systems are all together expressed as one operator. If I and J are N-bit numbers then $I <, =, \neq, > const$ and $I <, =, \neq, > J$ require only one operator. The operators can be 'controlled' by setting values to a subset of bits, then the remaining bits of the solutions can be output in linear time.

3. **CNF-solver. Main features:**
   Complete; Very small memory requirements; Memory is strictly constant during the run including the case when all solutions are sought; Record performance for crossover-point hardest problems; It is an optimal replacement for Davies-Putnam-Loveland types of programs.

4. **General-purpose Conjunctive-Form solver. Main features:**
   Accepts data in the form of operators with 5 productions; Runs pigeon-hole benchmarks in absolutely record time (6 sec for hole10, 1900 sec for hole14 on Pentium Pro 180, 40 MB RAM); May be used for the problems of the asynchronous design type, planner applications etc.; Can be adjusted for other types of problems.

5. **Application areas:**
   Knowledge learning: Discovery af approximate logical functions from a set of examples. This can be also used in design verification of IC's. Electronics design - one can suggest entirely new designs of IC's. Planning, scheduling, resource allocation. Graph-colouring type problems. Any problem where local search is used: One writes down the problem as a collection of positional operators and performs a local search on this very compact representation. An algorithm to solve nonlinear systems in GF(2) of arbitrary degree was developed and will be implemented shortly. This opens up a very broad application field. Our research and development process provides us with valuable insights into the interesting boundary between the P and NP-complete problems and we hope will clarify this central problem of modern discrete mathematics.

****

END OF E-MAIL QUOTATION