ASSIP-T. A THEOREM PROVING MACHINE

Werner Dilger
Fidunhofer-Institut fur Informiations- und Datonverarbeitung
D-7500 Karlsruhe 1

Hans-Albert Schneider
Computer Science Department
University of Kaiserslautern
Postfach 3049
D-6750 Kaisorslautern

ABSTRACT

An associative processor for theorem proving in first order logic is described. It. is designed on the basis of the deduction plan method, introduced by *Cox* and Pietrzykowski . The main features of this method are the separation of unification from deduction and the incorporation of a method for intelligent backtracking- This kind of backtracking is based on a special unification procedure. An improved version of this unitization procedure is given, which outputs a unification graph with constraints. In the case of a unification conflict, sufficient inf:onnation for a directed backtracking step can be gained from the unifiestion graph. According to the deduction plan method, the ASSIP-T memory consists of two parts, one for the deduction plan and the other for the unification graph. ASSIP-T can perform deduction and unification in parallel. Both memory parts consist of. a set of subparts each of which keeps the information about clauses or terms, respectively. A subpart is a 1inear array of cells provided with a control unit and can be regarded as a subprocessor.

1. In Production

The progress of microelectronics allows the realizations of more and more powerful processors for special purposes. One such type of processors is the associative processor. Its associative memory allows content oriented parallel access to the data stored in it. This makes the associative processors well suited for pattern handling processes. In artificial intelligence e.g., most processes are pattern directed deductions. One of it is theorem proving. In this paper a model of an associative processor is described which is able to prove theorems of first order logic. It is designed on the basis of the deduction plan method, i.e. it incorporates a method for intelligent backtracking.

After some basic definitions in the second section, the deduction plan method is described. The special unification procedure used within this method follows. The output of this procedure is a unification graph with constraints. In the case of a unification conflict, the unification graph gives sufficient information for a directed backtracking step. This is described in section 5. Then the structure of the ASSIP-T processor which is aimed to perform the deduction plan method is described. Section 7 gives the data structures which are to be mapped on the ASSIP-T memory. Finally, the representation of the data structures in the ASSIP-T memory is sketched.

2. Basic Definitions

A *labelled graph* is a triple $G = (V(G), I(G), E(G))$ where $V(G)$, $I(G)$, and $E(G)$ are the sets of nodes, labels, and edges respectively. A *path* of length n in G is a sequence $w = v_1, e_1, v_2, e_2, \ldots, e_n, v_{n+1}$ $(n \geq 0)$ with $v_i \in V(G)$ and $e_j \in E(G)$. If $v_1 = v_{n+1}$, the path is called *closed*. A closed path which contains each inner node at most once is called a *cycle*.

Assume there are given disjoint alphabets of *variables, function symbols* and *predicate symbols*. Each function and predicate symbol has an arity. A *constant* is a 0-ary function symbol. An *expression* is a variable or a term. A *term* is a constant or a string of the form $f(q_1, \ldots, q_n)$, where f is an n-ary function symbol $(n \geq 1)$ and $q_1, \ldots, q_n$ are expressions. An *atom* is a string of the form $P(q_1, \ldots, q_n)$, where P is an n-ary predicate symbol $(n \geq 0)$ and $q_1, \ldots, q_n$ are expressions. If A is an atom, then A and -A are *literals*. A *clause* is a finite set of literals. The empty clause is denoted by □.

A *constraint* is a set consisting of two expressions. A set of constraints is called a *constraint set*. If p and q are expressions (terms), then p is a *subexpression (subterm)* of q if p = q or $q = f(q_1, \ldots, q_n)$ and p is a subexpression (subterm) of one of the q_i. An expression (term) p is a subexpression (subterm) of a constraint set C, if there is a constraint $\{q_1, q_2\}$ in C such that p is a subexpression (subterm) of q_1 or of q_2. The set of all subexpressions of C is denoted by SEXPR(C).

A *substitution* is a finite set of pairs (v,q), denoted by v/q, where v is a variable and q an expression and $v \neq q$. *Application* of a substitution $\sigma = \{v_1/q_1, \dots, v_n/q_n\}$ to an expression or a literal p is the replacement of each occurrence of v_i in p by q_i, for all i = 1,...,n. σ is called a *renaming* if $q_1, \dots, q_n$ are pairwise different variables and $\{v_1, \dots, v_n\} \cap \{q_1, \dots, q_n\} = \emptyset$. A clause cl_1 is called a *variant* of a clause cl_2 if cl_1 and cl_2 have no variables in common and there is a renaming σ such that $cl_1 = \sigma cl_2$. If $E = \{p_1, \dots, p_m\}$ is a set of expressions then a substitution σ is called a *unifier* of E, if $\sigma p_1 = \dots = \sigma p_m$. E is then called *unifiable*. σ is called a *most general unifier* of E if for each unifier τ there is a unifier ρ such that $\tau = \sigma \cdot \rho$.

Let $C = \{c_1, \dots, c_n\}$ be a constraint set. The set BE(C) of *Boolean expressions* over C is defined by

1. $0, 1, c_1, \dots, c_n \in BE(C)$.
2. If $B_1, B_2 \in BE(C)$, then $(B_1 \vee B_2)$, $(B_1 \wedge B_2) \in BE(C)$.
3. BE(C) contains no other elements.

3. Deduction Plans

The deduction plan method is a resolution based method, i.e. a refutation method. It starts with a set of clauses and tries to construct a "closed" and "correct" deduction plan. If it succeeds, the clause set is proved to be unsatisfiable. The central idea of the method is to separate deduction from unification. This allows the application of a special unification algorithm which, in the case of a unification conflict, not simply stops with failure, rather it yields information about the causes of unification conflicts, namely certain deduction steps, which then can be reset. In section 5 this way of processing is called "intelligent backtracking".

The nodes of the deduction plan are the input clauses and eventually variants of them. Two clauses can be connected by an edge if they contain literals with the same precidate symbol but different signs (negated or not negated). Therefore a (labelled) edge between two clauses cl and cl_2 is a triple (cl_1 (t,u,v) ,cl_2,) , where u and v are literals in cl_1 and cl_2 respectively, satisfying the condition on their predicate symbols and negation signs, t is the type of the edge. There are two types of edges: SUB and RED. All edges are of type SUB except those referring backward to a clause which is already in use. If each literal in each clause included in the plan occurs in an edge, the deduction plan is closed. If the set of pairs of torms arising from the pairing of literals by edges is uniliable, the deduction plan is correct. Cf. for this section (Cox and Pietrzykowski 1979) and (Cox and Pietrzykowski 1981).

Definition

Let S be a set of input clauses and $L = \bigcup_{cl \in S} cl$. A *deduction graph* on S is a graph G = (V(G),I(G),E(G)) which has the variants of S as node set V(G), $I(G) \subseteq \{SUB,RED\} \times L \times L$ with: if $e = (cl_1, b, cl_2) \in E(G)$ then b = (t,u,v), $u \in cl_1$, $v \in cl_2$. t is called the *type* of the edge e, a the *starting literal* and v the *target literal.* A literal u of a clause cl is called *key literal* iff there is an incoming edge with type SUB and target literal u. Each literal u of a clause cl is called a *sub problem* iff it is not a key literal. A subproblem u E cl is *open* iff there is no outcoming edge with starting literal u. A subproblem u is called *closed* iff it is not open. os(G) is the set of open subproblems of a deduction graph G. G is called *closed,* iff os(G) - 0.

A node cl is called *predecessor* of a node cl., iff there is a path from cl to cl_0 which contains only edges of type SUB *(SUB-path).* If u is the starting literal of the first edge of a SUB-path from cl to cl_2, then u is called *preceding literal* of cl_2 and cl_2 is called *successor* of c1.

We omit the definition of the deduction plan here. It is a deduction graph which is constructed by a number of deduction steps, i.e. edge drawing steps, starting from a basic plan which consists of one node only.

Example

S = {{P(x), Q(y), R(f(x,y))},
{-P(g(x)), V(x)},
{-P(g(x)), -V(x)},
{-Q(x), S(x), -T(x)},
{-S(a)},
{-S(b)},
{T(b)},
{-R(x)}}

is a set of eight input clauses. Figure 1 shows a closed deduction plan for S. The edges are drawn in such a way that they begin beyond the starting literal and point to the target literal. Therefore they are only labelled by their type and, beyond it, by the numbers of the steps in the plan construction within which the edges were drawn. The literals $-P(g(x_2))$, $-V(x_3)$, $-Q(x_4)$, -S(a), T(b), and $-R(x_5)$ are key literals, the other literals are subproblems. The first clause in S is the basic node, it is a predecessor of all nodes.

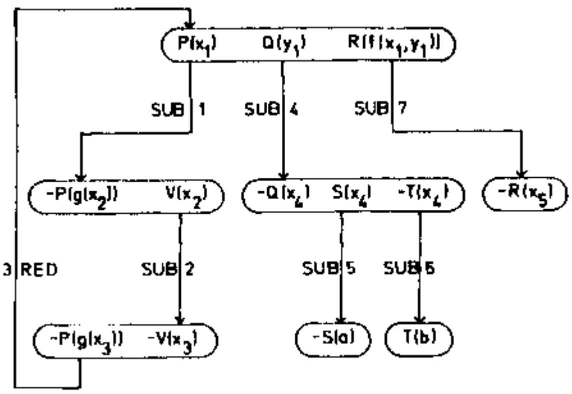

Figure 1: A closed deduction plan

Definition

Let G be a deduction plan and e an edge of G with label (t,u,v), where (omitting the sign) $u = P(u_1,\ldots,u_n)$, $v = P(v_1,\ldots,v_n)$ (n≥0). To e a constraint set C(e) is assigned by

$$C(e) = \{\{u_1,v_1\},\ldots,\{u_n,v_n\}\}$$

A constraint set C(G) is assigned to G by

$$C(G) = \bigcup_{e \in E(G)} C(e)$$

G is called *correct* iff C(G) is unifiable. θ(G) denotes the most general unifier of C(G). θ(G)os(G) is the clause *derived from* G. If G is closed, the clause derived from G is the empty clause.

Soundness and completeness of the deduction plan method are shown in the references given above.

4. Unification Graphs With Constraints

Unification by means of unification graphs with constraints is closely related to the unification method of Cox, (Cox 1981). It simplifies this method but is still sound and complete. Cf. for this section (Dilger and Janson 1983) and (Dilger and Janson 1984).

The unification process starts with a constraint set C. By two steps, the *transformation step* and the *sorting step*, it yields a *unification graph with constraints*, *UwC* for short, for C. A UwC consists of

- the node set V(UwC) = SEXPR(C)
- the label set L(UwC) = 2^C
- the edge set E(UwC) = EU(UwC) ∪ ED(UwC) where $EU(UwC) \subseteq V(UwC) \times (2^C - \{\emptyset\}) \times V(UwC)$ and $ED(UwC) \subseteq V(UwC) \times \{\emptyset\} \times V(UwC)$

EU(UwC) is a set of undirected edges, ED(UwC) a set of directed edges. Construction of UwC starts with the initial graph UwC_I which consists only of the nodes. EU(UwC) is determined in the transformation step, ED(UwC) in the sorting step.

Definition

A path in UwC which contains only edges from EU(UwC) is called a *connection*. A connection $v = p_1,e_1,\ldots,e_n,p_{n+1}$ is called *simple* iff $p_i \neq p_j$ for all i,j ($1 \le i < j \le n+1$). A closed path in UwC which contains at least one edge from ED(UwC) is called a *loop*. A loop is called *simple* iff $p_i \neq p_j$ for all i,j such that $1 \le i < j \le n+1$. If e = (p,a,q) is an edge in E(UwC), then a is called the value of e, denoted val(e) = a. Let $w = p_1,e_1,\ldots,e_n,p_{n+1}$ be a path in UwC. Then the value of w is

$$val(w) = \bigcup_{i=1}^{n} val(e_i)$$

The transformation step

The algorithm of the transformation step can be found in (Dilger and Janson 1984). It draws undirected edges between the nodes in the following way: If $c_i = \{p,q\}$ is a constraint, the nodes p and q are connected by the edge $e = (p,\{c_i\},q)$. This results in a (possibly empty) set of new constraints, which are treated later on in the same way.

Example

Let $C = \{c_1,c_2\}$ be a constraint set with

$c_1 = \{G(s,z),\ G(u,F(y,y))\}$

$c_2 = \{u,\ F(y,G(s,z))\}$

The initial UwC consists only of the nodes SEXPR(C) and is shown in figure 2. The first constraint c_1 is removed, an appropriate undirected edge is added to the UwC and the new constraints {s,u} and {z,F(y,y)} are added to the constraint set.

Now the second constraint is removed from the constraint set. Because u is a variable, there cannot be formed any new constraints, only an edge is added to the UwC. The remaining two constraints are treated as the second one. Because they had their origin in the first constraint,

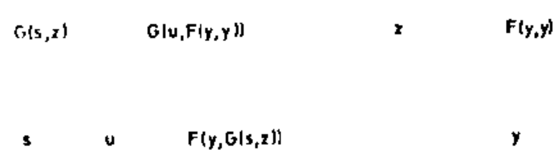

Figure 2: The initial UwC for the constraint set C

the edges in the UwC arc labelled by {c }. At the end of the transformation step the UwC has the form represented in figure J.

The sorting step

The transformation step classifies the nodes of UwC in such a way that two nodes belong to the same class iff there is a connection between them. In the example *above* we have four classes. In the sorting step, first a graph U is constructed which consists of these classes as nodes and which has a directed edge labelled by f from class X to class Y iff there is a term f(p ,...,p) in X and an expression p , (i ∈ {1,...,n}) in Y.

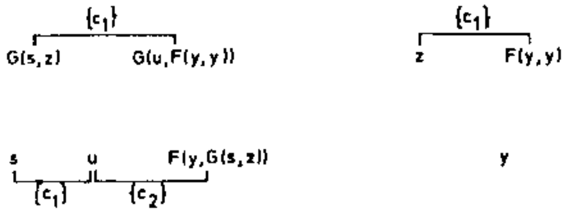

Figure 3: The UwC at the end of the transformation step

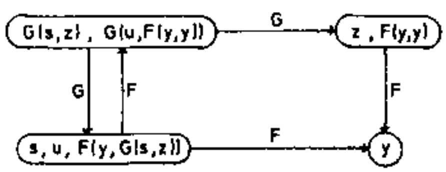

Figure 4: The graph U for the UwC

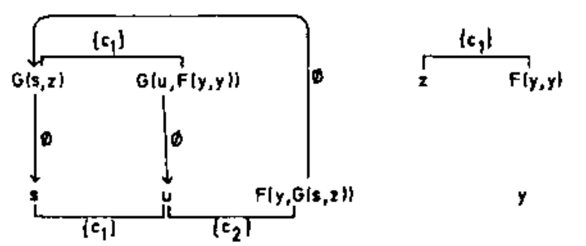

Figure 5: The complete UwC at the end of the sorting step

This graph is shown for the example in figure 4. Now the edges of U which belong to a cycle are added to the UwC as edges between the appropriate nodes and labelled by 0. So we get the complete UwC of figure 5.

Soundness and completeness of the unification algorithm are proved in (Dilger and Janson 1984). The main theorem is: A constraint set C is unifiable iff all terms in UwC which are connected by a simple connection begin with the same function symbol and UwC contains no simple loops.

Thus, e.g., our example constraint set is not unifiable because the UwC of figure 5) contains a simple loop.

S. Intelligcnt Backtracking

If during the unification process a unification conflict has been detected, i.e. a clash (unification of terms with different function symbols) or a cycle, the actual deduction plan is not correct. One or several steps in the construction have to be reset in order to get a correct plan. By means of the information kept by the UwC these steps can be determined immediately. The numbers of the deduction steps are contained in the labels of the undirected edges of UwC. Therefore, we have to examine the values of certain paths through UwC. First, the relevant values are gathered in the sets ATTACH and LOOP.

ATTACH := {a ⊆ C | a is the value of a simple connection in UwC between terms p and q with different function symbols}

LOOP := {a ⊆ C | a is the value of a simple loop in UwC}

We define:

$$B_{ATTACH} := \prod_{a \in ATTACH} \sum_{c \in a} c$$

$$B_{LOOP} := \prod_{a \in LOOP} \sum_{c \in a} c$$

$$B_{UNIF} := B_{ATTACH} \wedge B_{LOOP}$$

The minimal disjunctive normal form of B_{UNIF} has the form

$$B'_{UNIF} = B_1 \vee \ldots \vee B_k$$

for some k≥1, where each B_i is a conjunctive term. From B'_{UNIF} the minimal conflict sets are determined by

$$mcs_i := \bigcup_{c \text{ occurs in } B_i} \{c\} \quad (i = 1,\ldots,k)$$

For details cf. (Dilger and Janson 1984).

Example

Consider the deduction plan of section 3, represented in figure 1. Following the edges according to their numbers we get the constraints

1: $\{x_1, g(x_2)\}$
2: $\{x_2, x_3\}$
3: $\{g(x_3), x_1\}$
4: $\{y_1, x_4\}$
5: $\{x_4, a\}$
6: $\{x_4, b\}$
7: $\{f(x_1, y_1), x_5\}$

The UwC for these constraints is shown in figure 6. It has no directed edges, because the graph U, constructed in the sorting step, contains no cycles.

There is a clash in the UwC, namely a simple connection between a and b. Therefore, ATTACH = {{5,6}}. Clearly, LOOP = ∅. Thus, $B_{ATTACH} = 5 \vee 6$, $B_{LOOP} = 1$, $B_{UNIF} = 5 \vee 6 = B'_{UNIF}$ and $mcs_1 = \{5\}$, $mcs_2 = \{6\}$.

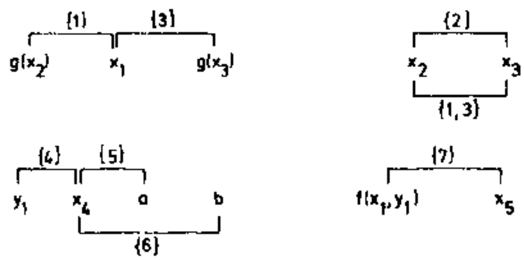

Figure 6: A complete UwC

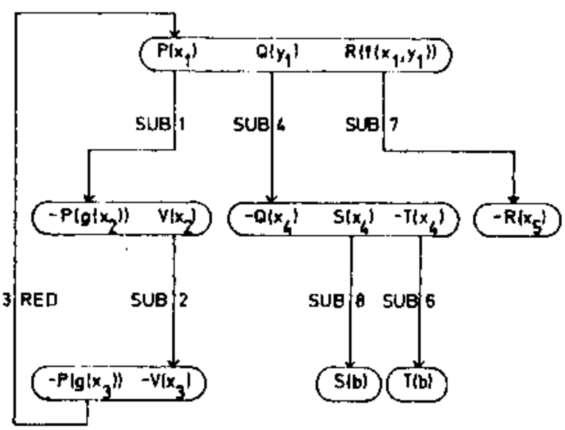

Figure 7: A closed correct deduction plan

Backtracking is performed as follows. Take for the backtracking step mes = {S} Edge number fj and node -S(a) are removed from the plan. Thereby, the literal S(x.) be comes an open subprobleni. But there is another clause in the input clause set which fits so close the literal, namely {-S(b)}. This yields the closed correct deduction plan of figure 7. Thn reader is invited to check that backtracking with .ncs₂ - {6} does not result in a c1osed plan .

6. The structure Of ASSIP-T.

In che deduction plan method, deduction and unification are separated from each other. For deduction, the data structure "doduction plan " is used, for unificdtion the data structure "utiification

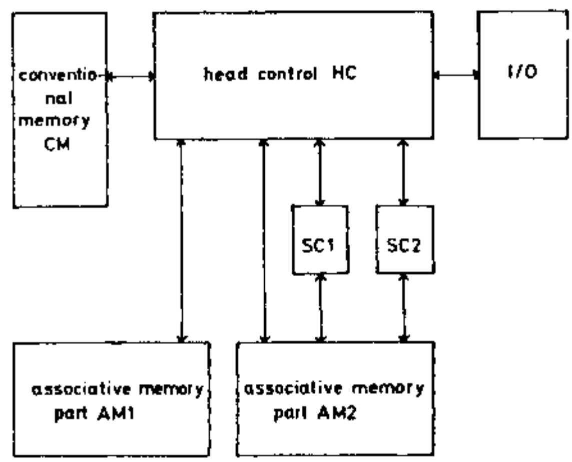

Figure 8 : The structure of ASSIP-T

gr,iph wiih constrainLs". In ASS1P-T, both are kept in appropriate pΛrts of the associative meinory. Thus, the associative memory is divided in two main parts, AMI for the deduction plan and AM2 for the UwC, of. figure 0. The control unit of the processor consists of fo11r components:

- the head control HC
- two subcont.rols SC1 and SC2
- a conventiorial memory CM

The subcontrols operate on the UwC. They t-an work independently froin each ot.her, but under control of HC, so they can work in parallel and this is useful during the initial construction of the UwC and during its reconstruction after a backtracking step. Thus, we have not only parallel access to the data in the associative memories, rather there are two further steps to parallel processing: one by the parallel treatment of deduetion plan and UwC, the other by the use of SC1 and SC2 in parallel. For details cf. (Dilger and Schneider 1985). For an introduction to and a survey on the field of associative processors cf. (Fu and Ichikawa 1982), (Kohonen 1984), (Parhami 1973) and (You and Fung 1975).

7. Associative Memory Oriented Data Structures

Several data structures are used for an associative memory oriented representation of deduction plans and UwCs, i.e. representations that can easily be mapped on associative memories. The design principles for the data structures are:

1. Deduction plan and UwC - both being graphs - are taken as sets of nodes together with their edges.
2. A clause together with all its variants is represented as only one data object and therein their constant part is represented only once. The same holds for the terms of the UwC.

Due to lack of space we omit the data structures, which can be found in (Dilger and Schneider 1985), and only give some idea of them.

For each literal, we keep in its variants-part the edges to other literals, represented by the target literals and the edge labels, because in fact they are drawn between variants of clauses. This way of storing edges can be thought to be similar to the way they are drawn e.g. in figure 1.

The nodes of the UwC are variants of expressions. Therefore, we store all expressions which are variants of one another in the same part of AM2 (cf. section 8) together with the edges incident of them.

Because we build variants by just indexing the variables (cf. figure 1), we are able to represent the information "edge e is incident to node t" by simply storing the index of t's variables at e, too. Storing directed edges is done in a most efficient way, which just needs one bit for each argument of the respective term.

8. Representation And Handling Of The Data Structures In The ASSIP-T Memory

We will sketch here the representation of the UwC in the memory part AM2. It is similar for the deduction plan. AM2 is divided into several parts, one for each object of type EXPRESSION (that is, an expression, its variants and the edges incident to them). Every AM2-part consists of a linear array of cells and is provided with a special control, called the "EXP-control". The entries in an object of type EXPRESSION can all be represented by the data types INTEGER and BOOLEAN. Therefore all cells of the AM2-parts have the same form. They consist of

- a logical unit
- a control bit
- a 4 bit flag register
- a 32 bit data register

cf. figure 9. The purpose of the flag register is to characterize the type of information which actually is stored in the data register, e.g. index and class of variants, information about edges etc. Thus, each cell can store an arbitrary part of an EXPRESSION.

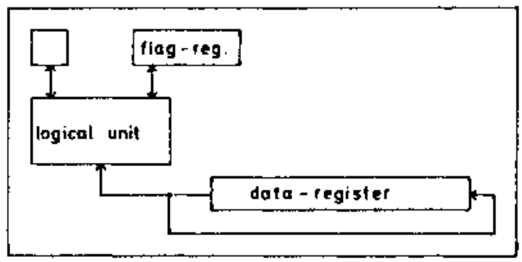

Figure 9: A subprocessor cell of ASSIP-T

The EXP-control has to perform entry, change and query instructions on the components of an object of type EXPRESSION. The head control on the other hand just has to broadcast information to the AM2-parts and to gather it from them by means of instructions like

"FOR_ALL <expression> WITH <condition>
DO <instruction>"

or "FOR_ONE <expression> WITH <condition>
DO <instruction>"

Thus, the two-level organisation of the ASSIP-T memory corresponds to a two-level evaluation of the instructions.

9. Conclusion

As far as we know there is no other approach similar to ours. The architecture of the fifth generation inference machine is data flow oriented and does not take into consideration associative access to data cf. (Moto-oka and Fuchi 1983). The main problem with our approach is the storage of the unification graph because its number of edges has an upper bound that is exponential with respect to the number of deduction steps. One may assume that this upper bound will never be reached in practice, but we have to work out another representation of the edges. By means of several head control-subcontrol-groups, we should be able to perform 0R-parallel as well as AND-parallel processing due to the separation of deduction and unification.

REFERENCES

Cox, P.T. On determining the causes of nonunifiability. Auckland Computer Science Report No 23, University of Auckland, 1901.

Cox, P.T. and Pietrzykowski, T. Deduction plans: A basis for intelligent backtracking. University of Waterloo Res. Rep. CS-79-41, 1979.

Cox, P.T. and Pietrzykowski, T. Deduction plans: A basis for intelligent backtracking. IEEE Trans. Pattern Analysis and Machine Intelligence, vol. PAMI-3, (1) 1981, 52-65.

Dilger, W. and Janson, A. Unifikationsgraphen als Grundlage fur intelligerites Backtracking. Proc. of the German Workshop on Artificial Intelligence, Informatik-Fachberichte 76, Springer-VerJag, 1983, 189-196.

Dilger, W. and Janson, A. A unification graph with constraints for intelligent backtracking in deduction systems. Interner Bericht 100/84, Fachbereich Informatik, Universitat Kaiserslautern, 1984.

Dilger, W. and Schneider, II.-A. A theorem proving associative processor, In preparations .

Fu, K.S. and Ichikawa, T. (eds) Special computer architectures for pattern processing. CRC Press, Boca Raton, Florida," 1982.

Kohonen, T. Self"-organizations and associative memory. Springer, Berlin, 1984.

Moto-oka, T. and Fuchi, K. The architectures in the fifth generation computers. Proc. of the IFIP 83, 1983, 589-602.

Parhami, B. Associative memories and processors: An overview and selected bibliography. Proc. of the IEEE 61, 1973, 722-730.

You, S.S. and Fung, H.S. Associative processor architecture - a survey. Proc. of the Sagamore Computer Conference 1975.