Representation of Mathematical Knowledge

J. Calmet and I.A. Tjandra
University of Karlsruhe; Department of Computer Science
Institut für Algorithmen und Kognitive Systeme
Am Fasanengarten 5; W-7500 Karlsruhe; Germany

Abstract

We describe a method to build an environment for processing mathematical domains of computation. Basically, the environment may aid the user in: (i) specifying correct abstract computational structures and their models, (ii) completing the sets of properties of operators by means of a learning method and (iii) determining the correct domain in which a mathematical computation must be performed. The system is built upon a hybrid knowledge representation system.

keywords: expert system, mathematical domains of computation, knowledge bases.

1 Introduction

We have designed a shell for knowledge representation which can be used as an environment capable of representing arbitrary mathematical domains linked by algebraic relationships, and of performing correct computations on these domains. A correct representation of domains is achieved through a constructive definition taking into account the functions of the domains and the properties. This includes consistency, completeness of properties and correctness of function implementations with respect to these properties.

The environment (Figure 1) is built upon a hybrid knowledge representation system, called MANTRA [Bit]. MANTRA was designed upon the following two principles: (i) several cooperating formalisms are better then a unique representation formalism, and (ii) a clear unified semantics explaining the meaning of the knowledge representation language is fundamental.

The specification component is used to specify and to represent arbitrary abstract computational structures and their domains. This component aids the user in building an ACS, also based on other known "lower" ACS, taking into account the consistency of the set of properties.

The learning component acts as an automatic knowledge acquisition tool in order to get a complete set of properties. This is performed by investigating the results of computations

of the method of explanation-based generalization.

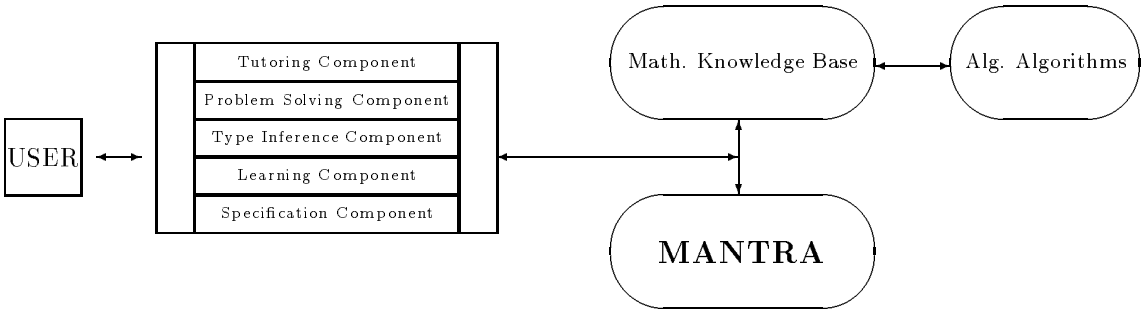

Figure 1: The structure of the environment

The paper is organized as follows. In section 2 we describe MANTRA. In section 3 we present a formal definition of signatures, domains, terms and interpretations. Section 4 deals with abstract computational structures, domains and their representation. Section 5 deals with the method adopted to complete a set of properties. Finally, we present some concluding remarks on the design of the environment in section 6.

2 MANTRA

MANTRA (Modular Assertional, Semantic Network and Terminological Representation Approach) is a shell for knowledge representation. It is best described in [Bit] where a list of references is given. The main characteristics of MANTRA are: (i) The introduction of a multilevel architecture for hybrid systems together with a methodology to define a unified semantics for knowledge representation methods and their interaction. (ii) The integration of two features which are usually not found in hybrid systems: Inheritance with exception and heuristic programming. (iii) The extension of the Frame terminological language to accept n-valued relations instead of binary roles only. (iv) The semantic definition of non-monotonic inheritance with exceptions using the four-valued logic approach. (v) All algorithms for inference procedures are decidable. (vi) High interaction among the different knowledge representations covered by the system.

Logic, frames and semantic nets are used to express declarative knowledge and are called epistemological methods because they are associated to models in the external world. A production system represents a complete programming paradigm including the procedural aspect and is called a heuristic method because of this feature. One or more inference procedures are associated to each of the epistemological methods.

First order logic offers a very powerful inference procedure allowing to generate all of the entailed (i.e. implicitly represented) knowledge by a given amount of explicitly represented knowledge. The problem of verifying if a given element of knowledge is entailed by a certain amount of explicit knowledge is, however, not decidable in first order logic. To avoid this problem we adopt a non-complete inference procedure and a query procedure

These procedures are semantically defined through a four-valued logic approach [Be].

Our frame method is a terminological language as in Brachman [Br et al.] But it extends its capabilities by accepting n-valued relations. The main inference procedure is the subsumption procedure. A concept or relation subsumes another one if the former includes the latter.

Semantic networks are methods based on the abstraction of nodes and edges. The method adopted in our approach allows to define hierarchies with exception. The inheritance procedure is the main inference procedure available in this method. It allows to verify if a class is a subclass of another one given an explicit hierarchy consisting of either default or exception links. The so-called sceptical inheritance approach [Ho et al.] has been selected within, again, the four-valued semantic approach.

These three epistemological methods are not very powerful when standing alone because of the adoption of the four-valued semantics which weaken the deductive power of the system. But, this choice was mandatory in order to have only decidable inference algorithms and a semantically sound system. The deductive power is, however, very much increased by the association of the three methods. This association is performed through the definition of hybrid inference procedures allowing to generate new knowledges from those acquired through the three methods separately. This interaction is defined not by the inference procedures themselves but by the semantic specification of the results that these procedures should provide. The architecture of MANTRA is defined along three levels: the epistemological level, the logical level and the heuristic level as shown in figure 2.

The epistemological level includes the three methods described above and it is extendible. Each of the corresponding modules is thus constructed around some epistemological notions: predicates, functions and constants in the logic module, concepts and relations in the frames module, classes and hierarchies in the semantic net module. Syntactically, each one of these modules consists of a set of primitives which are used to manipulate the epistemological notions. The four-valued semantics enables to model ignorance and inconsistency and thus provides a semantics for an incomplete inference mechanism.

The second level introduces the concept of knowledge base. Two primitives are used to store facts and to interrogate knowledge bases respectively. There are eight possible interactions among modules: logic-frame, logic-semantic networks (SN), frame-SN. SN-logic, SN-frame, logic-frame-SN, frame-logic-SN and SN-logic-frame. All these interactions have been semantically defined and algorithms for the first three have been proposed and proven sound and complete with respect to the semantic definition.

The third level is the heuristic level. It consists of primitives allowing the definition of production systems. At this level the conflict resolution and flow control strategies applied during the execution of these production systems can be explicitly chosen through special primitives.

The common basis for all the implemented inference procedures is the unification function. A specialized unification package has been implemented in KCL. It is based upon the almost linear algorithm of Martelli and Montanari [Ma,Mo].

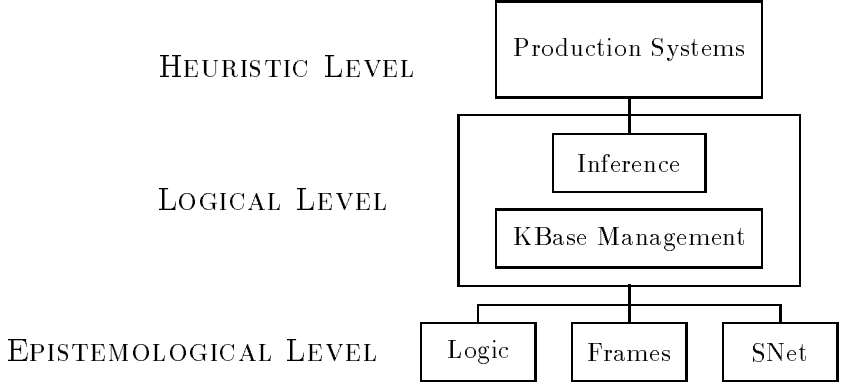

Figure 2: The architecture of MANTRA

3 Basic Notions and Notations

$\Sigma = \langle \mathcal{S}, \Omega \rangle$ is an order-sorted signature consisting of $\mathcal{S}$, a finite set of order-sorted sorts, and Ω, the set of operator symbols. Ω is the union of pairwise disjoint subsets of $\Omega_{w,s}$, the sets of operator symbols with argument sorts $w \in \mathcal{S}^*$, i.e. $w = s_1 \times \cdots \times s_n$ for $n \geq 0$ and $s_i \in \mathcal{S}$, and range sort $s \in \mathcal{S}$.

$\|w\|$ and λ are used to denote the length of w and the empty word respectively. C_s is the set of constant symbols of sort s $(\Omega_{\lambda,s})$. The ordering $\sqsubseteq$ is extended to words by $s_1 \times \cdots \times s_n \sqsubseteq s_1^{'} \times \cdots \times s_n^{'}$ iff $\forall i = 1, \cdots, n : s_i \sqsubseteq s_i^{'}$.

A domain $\mathcal{D}$ with the signature $\Sigma = \langle \mathcal{S}, \Omega \rangle$, also called Σ-domain, is a pair $\mathcal{D} = \langle \mathcal{U}, \mathcal{F} \rangle$ such that $\mathcal{U}$ is an $\mathcal{S}$-indexed family of sets and $\mathcal{F}$ is a Ω-indexed family of functions with:

(i) $\mathcal{U}_s$ are sets for all $s \in \mathcal{S}$, called the universe of sorts s in the domain.

(ii) $\mathcal{F}_f \in \mathcal{U}_s$ if $f \in \Omega_{\lambda,s}$ and $s \in \mathcal{S}$, called constants of $\mathcal{D}$.

(iii) $\mathcal{F}_f \in \mathcal{U}_{s_1} \times \cdots \times \mathcal{U}_{s_n} \rightarrow \mathcal{U}_s$, functions for all operator symbols f, if $f \in \Omega_{s_1 \times \cdots \times s_n, s}$ with $n \geq 1$, $s, s_i \in \mathcal{S}$ and $i = 1, \cdots, n$.

Let $\Sigma = \langle \mathcal{S}, \Omega \rangle$ be a signature and X_s for each $s \in \mathcal{S}$ a set of variables of sort s. We assume that the variables of distinct sorts are distinct and that no variable is a member of Ω. The union $X = \cup_{s \in \mathcal{S}} X_s$ is called the set of variables with respect to Ω.

The sets $T_{s,\Omega}(X)$ of terms of sort s are inductively defined as follows:

(i) $X_s \cup C_s \subseteq T_{s,\Omega}(X)$ (*basic terms*),

(ii) $f(t_1, \cdots, t_n) \in T_{s,\Omega}(X)$ (*composite terms*) for all operator symbols $f \in \Omega_{s_1 \times \cdots \times s_n, s}$ and all terms $t_i \in T_{s_i,\Omega}(X)$ $i = 1, \cdots, n$,

variables of sort s, also called *ground terms* of sort s, is defined for the empty set $X = \emptyset$ of variables as follows:

(iv) $T_{s,\Omega} = T_{s,\Omega}(\emptyset)$. The set of terms $T_\Sigma(X)$ and the set of terms without variables T_Σ are defined as follows:

(v) $T_\Sigma(X) = \bigcup_{s \in \mathcal{S}} T_{s,\Omega}(X)$ with $s_1, s_2 \in \mathcal{S},\ s_1 \sqsubseteq s_2,\ t \in T_{s_1,\Omega}(X) \Rightarrow t \in T_{s_2,\Omega}(X)$ and $T_\Sigma = \bigcup_{s \in \mathcal{S}} T_{s,\Omega}$ with $s_1, s_2 \in \mathcal{S},\ s_1 \sqsubseteq s_2,\ t \in T_{s_1,\Omega} \Rightarrow t \in T_{s_2,\Omega}$. This also implies that each sort (type) inherits all terms possessed by its subsorts (types).

Let T_Σ be the set of terms of signature $\Sigma = \langle \mathcal{S}, \Omega \rangle$ and $\mathcal{D}$ a Σ-domain. The *interpretation* $\varphi : T_\Sigma \to \mathcal{D}$ is recursively defined as follows:

(i) $\varphi(f) = \mathcal{F}_f$ for all $f \in \Omega_{\lambda,s},\ s \in \mathcal{S}$.

(ii) $\varphi(f(t_1, \cdots, t_n)) = \mathcal{F}_f(\varphi(t_1), \cdots, \varphi(t_n))$ for all $f(t_1, \cdots, t_n) \in T_\Sigma$.

Given a set of variables X for $\Sigma = \langle \mathcal{S}, \Omega \rangle$, a Σ-domain $\mathcal{D}$ and an assignment $\psi : X \to \mathcal{D}$ with $\psi(x) \in \mathcal{U}_s$ for $x \in X_s$ and $s \in \mathcal{S}$, the *extension* $\xi : T_\Sigma(X) \to \mathcal{D}$ of the assignment $\psi : X \to \mathcal{D}$ is recursively defined as follows:

(i) $\xi(x) = \psi(x)$ for all variables $x \in X$.
$\xi(f) = \mathcal{F}_f$ for all $f \in \Omega_{\lambda,s},\ s \in \mathcal{S}$.

(ii) $\xi(f(t_1, \cdots, t_n)) = \mathcal{F}_f(\xi(t_1), \cdots, \xi(t_n))$ for all $f(t_1, \cdots, t_n) \in T_\Sigma(X)$

4 Abstract Computational Structures

Initially, we have to extend the notion of signature by adding some properties. Given a signature $\Sigma = \langle \mathcal{S}, \Omega \rangle$ and variables X with respect to Σ. A triple $p = \langle X, L, R \rangle$ with $L, R \in T_{s,\Omega}(X)$ for some $s \in \mathcal{S}$ is called a property of sort s with respect to Σ. The property $p = \langle X, L, R \rangle$ is said to be *valid* in a Σ-domain $\mathcal{D}$ if for all assignment $\psi : X \to \mathcal{D}$ we have $\xi(L) = \xi(R)$. *Ground properties* are properties $p = \langle X, L, R \rangle$ with $X = \emptyset$. In this case L and R are ground terms.

An *abstract computational structure* $ACS = \langle \Sigma, \mathcal{P} \rangle$ consists of a signature $\Sigma = \langle \mathcal{S}, \Omega \rangle$ and $\mathcal{P}$, a set of properties with respect to Σ. Based on other known ACS we can construct a new ACS by integrating all operators and properties possessed by the known ACS and by adding additional operators and properties into the new one. This implies that each ACS inherits all operators and properties possessed by the ACS on which it is based.

In order to represent such ACS by means of the knowledge representation formalisms provided by MANTRA we divide the construction of an ACS into the following parts: (i) ACS descriptor, (ii) ACS operators, (iii) ACS initial properties and (iv) ACS learned properties (see the example in figure 3).

can represent an ACS as a knowledge base that we call a *knowledge base module*. These modules, embodying the ACS and their models under consideration, are joined together into a semantic network preserving the relations among the ACS by means of the defined hierarchies. In the semantic network each node corresponds to a module and the links specify which entities are inherited by which modules within a particular hierarchy.

An ACS descriptor is represented by a concept, using the **Tell** primitive provided by the knowledge base management at the logical level, possessing the following relations: (i) ACS-id, a unique identifier representing the name of the ACS, (ii) ACS-mode, a symbol representing $\mathcal{S}$, (iii) parameters, a list of other (known) ACS representing $\mathcal{S}$ and (iv) based-on, a list of other (known) ACS on which the ACS is based.

Similarly, each operator of an ACS is represented by means of a concept possessing the following relations: (i) operator-id, a symbol representing the name of an operator, (ii) domain, a list of other known ACS, which are elements of $\mathcal{S}$, constituting the domain sorts of the operator, (iii) range, the range sort of the operator.

Properties are specified into two parts: Initial properties and learned properties. The representation of both parts is the same. The initial properties are the basic properties to be possessed by the operators. The learned properties remain to be acquired in order to complete the set of properties. In the next section we deal with a method for completing a set of properties.

A property $p = \langle X, L, R \rangle$ is represented by means of a logic formula, using the assertional module at the logical level, where each variable in X is bounded by a universal quantifier; L and R are treated as left-hand side and right-hand side respectively.

In order to construct a domain with respect to a particular ACS we have to implement each relevant function through an operator symbol specified in the ACS. A domain consists of three parts: (i) domain descriptor, (ii) function descriptors and (ii) the implementation of each function. The implementation of functions is supported by an embedded Lisp programming environment and relies on the classical algebraic algorithms.

5 Completing a set of properties

The next figure shows an ACS for group with all inherited operators and properties.

ACS group $\equiv$ **mode** g ;

based-on {} ;

function

f : (g,g) $\longrightarrow$ g
inv : (g) $\longrightarrow$ g
ne : () $\longrightarrow$ g ;

initial properties

p_0: $\forall$ x $\in$ g : f(ne,x) = x
p_1: $\forall$ x $\in$ g : f(inv(x),x) = ne

learned properties

p_3: *inv(ne) = ne*
p_4: $\forall\ x \in g$: *f(x,ne) = x*
p_5: $\forall\ x \in g$: *inv(inv(x)) = x*
p_6: $\forall\ x \in g$: *f(x,inv(x)) = ne*
p_7: $\forall\ x,y \in g$: *f(inv(x),f(x,y)) = y*
p_8: $\forall\ x,y \in g$: *f(x,f(inv(x),y)) = y*
p_9: $\forall\ x,y \in g$: *inv(f(x,y)) = f(inv(y),inv(x));*

end-of-ACS;

figure 3: Example

To complete the set of properties one can not rely on the Knuth-Bendix algorithm [Kn,Be] which is both inefficient and unpracticable for our purposes [Bi].

Let us suppose that we have an ACS for groups as defined above and we assume that the learned properties have been acquired. Let t_0 = f(f(inv(f(y,f(inv(y),x))),x),f(ne,x)) and t_{nf} = x, $t_0, t_{nf} \in T_\Sigma(X)$, be terms with respect to Σ. Since t_0 and t_{nf} belong to the same congruence class, the interpretations of both terms yield the same result, i.e. $t_{nf} =_R t_0$. Therefore, it is conceivable to reduce a term to its canonical normal form before interpreting it. This reduction process is accomplished syntactically. In order to achieve t_{nf} from t_0 the following reduction chain is processed.

$$t_0 \longrightarrow^{p_8} t_1 \longrightarrow^{p_1} t_2 \longrightarrow^{p_0} t_3 \longrightarrow^{p_0} t_{nf}$$

The underlying idea of the reduction of a term is that the problem of the validity of an equation in an ACS can be solved by the study of a congruence equation modulo a relation in the domain ($=_R$), a syntactic problem. In order to determine whether or not an equation is valid in the given ACS, i.e., whether or not two terms belong to the same congruence class, initially, we have to construct a **"complete"** set of properties.

Therefore, *completion* means according to [Ba et al.] acquiring sufficiently many properties such that, in particular, any application of a property in a proof within the equational theory can be transformed into a reduction proof.

We just keep the properties defined by the user as they are, i.e., at the beginning we do not try to make the set of properties complete. Instead, the interpretations of terms should be used as positive training instances from which new properties could be acquired incrementally. The new acquired properties constitute the *learned properties part* as mentioned above. This can be achieved by introducing the following basic steps: (i) Before t $\in T_\Sigma$ is interpreted, it should be reduced using the existing properties. In this step, we should use heuristics to support or to accelerate the reduction process. (ii) Interpret t. (iii) Try to find a relation between the result and the term which has been interpreted by using the method of goal regression [Wa]. This involves traversing their parse trees in the top-down manner and replacing constants by variables consistently.

form *"generalized term = generalized result"* . According to the notion of completion, as mentioned above, the generalized equation remains to be proven in order to acquire new properties. (v) Integrate the new acquired properties into the learned properties part.

Restricting the properties to the following forms: (i) The left hand side of a property can be reduced by no property but itself and (ii) The right hand side of a property can not be reduced by a property, we are capable of determining a method which can be applied in solving the problems in steps (iii) and (iv).

The problem occurring in step (iii) can be stated as follows: **given:** A term L and a term R, the result of the interpretation of L ($L, R \in T_\Sigma$), **determine:** $L' = R'$ $L', R' \in T_\Sigma(X)$, the generalization of the equation $L = R$.

The following basic steps, based on a depth-first tree traversal, describe the process of determining the generalization of $L = R$:

```
Build the parse tree of L according to the
corresponding context-free grammar [Ho,Ul] of the ACS;
Let Θ be a set of substitutions σ_i;
Θ = {σ_i | i = 1, ..., n};
Initialize Θ := {} ; n := 0 ;
for-each 0-ary function ''fn:() → val'' belonging to the
function implementation of the given domain
do
    σ_n := (fn/val);
    Θ := Θ ∪ {σ_n};
    n := n + 1;
traverse the parse tree of L in a depth-first manner;
for-each terminal node α being traversed do
    if ∃i : (σ_i ∈ Θ) ∧ (σ_i = (X/α))
        then substitute the terminal node using σ_i
        else
            compute an appropriate σ_n = (X',α)
            where X' is a variable not occurring in Θ;
            Θ := Θ ∪ {σ_n};
            apply σ_n on the terminal node ;
            n := n + 1;
if Θ can substitute all terminal nodes of R
    then return L' = Θ(R) else fail
```

The problem occurring in step (iv) can be stated as follows : **given:** An equation $L' = R'$ as hypothesis, **determine:** The proof of the given equation.

According to the notion of completion, as mentioned before, an equation can be eliminated if there is already a rewrite proof for it. Our problem in this step leads to acquiring properties, which can be used to prove the given equation.

To describe the algorithm, based on a breadth-first tree traversal, for acquiring such properties, we need two global variables: q of type Qu and the td (tree depth) of type integer, where Qu is a queue of pairs (left,right), where left and right are parse trees of terms of the given ACS.

the given equation $L' = R'$:

```
q := stock(empty-queue,(Parse-L', Parse-R'));
while not is-empty(q) do
    a := front(q);
    q := rest(q);
    breadth-first(a.left)
    if ( (the depth of a.left is equal to td) ∨
        (not match-property(a.left,a.right)) )
        then assert the property term(a.left)=term(a.right) to the learned properties
        part, where term(x) is the term corresponding to the parse tree of x
```

The procedure *breadth-first* traverses its input argument in a breadth-first manner and terminates at the tree depth where the first terminal node has been reached. After this termination, the global variable td is assigned to the depth of this terminal node.

The function *match-property* yields the value **true** if there exists a property $p_x = \langle X, L, R \rangle$ and a set of tree substitution S = $\{(\text{tree}_{i_l}/\text{tree}_{i_r})\}$ such that the following conditions hold:

1. Applying S to left-tree, where S is applied to each subtree of left-tree whose root is the node of left-tree at the depth td, yields a tree that is equal to the parse tree of L'.

2. a.right is equal to the parse tree of R'.

If there exists such S then each pair $(\text{tree}_{i_l}/\text{tree}_{i_r})$ is inserted into q, i.e., q := stock(q, $(\text{tree}_{i_l}/\text{tree}_{i_r})$). Otherwise the function *match-property* yields the value **false**.

6 Conclusion

We have investigated the possibility to use a hybrid knowledge representation to represent mathematical domains. The idea behind completing a set of properties is that a semantic problem can be solved by the study of an equation congruence modulo a relation, a syntactic problem. The currently existing completion methods have practical limitations due to the non-existence of a complete set of properties in many cases. We propose an approach to acquire the properties that can be used to prove a given equation. The major steps in acquiring such properties are the generalization and the proof of positive training instances. The proposed method is very reminiscent of the method of explanation-based learning [Mi et al.].

One of the further works in progress is to design the problem solving component. The main idea is that a, say mathematical or physical, problem can be solved by specifying the problem in the representation as defined in the environment and by using knowledge which has been stored in knowledge bases. Basically, it is very difficult to find a correct solution since, probably, the knowledge, which has been already stored, is not sufficient to solve the given problem. A conceivable approach consists in integrating an intelligent

an expert in order to acquire additional knowledge required to solve the problem.

To implement effectively the proposed approach we must still solve several problems, such as to find an approximated criterion to terminate the property acquisition procedure when, for instance, "sufficiently" many properties have been acquired (generally this problem is undecidable). To have an efficient implementation satisfactory running time must also be achieved. To find solution to such problems is the next step in this ongoing research project.

References

[Ba et al.] Bachmaier, L., Dershowitz, N., Hsiang, J. : *Proof Orderings for Equational Proof* ; Proc. LISC 86, 346 —357.

[Bi] Bibel, W. : *Automated Theorem Proving* ; Vieweg; 1982.

[Be] Belnap N.D.: *A Useful Four-Valued Logic*; in "Modern Uses of Multiple-Valued Logic", ed. G. Epstein and J.M. Dunn, on: Reidel, 1977, pp. 8 – 37.

[Bit] Bittencourt G.: *An Architecture for Hybrid Knowledge Representation*; Ph D Thesis; Universität Karlsruhe; 1990.

[Br et al.] Brachman R.J., Fikes R.E., Levesque H.J.: *KRYPTON: A Functional Approach to Knowledge Representation*; IEEE Computer, 16 (10), pp. 67 – 73, 1983.

[Ho et al.] Horty J.F., Thomason R.H., Touretzky D.S.: *A Skeptical Theory of Inheritance in Nonmonotonic Semantic Nets*; Report CMU-CS-87-175, Carnegie-Mellon Univ., 1987.

[Kn,Be] Knuth D.E., Bendix P.B. : *Simple Word Problems in Universal Algebras*; OXFORD, 263 —298 ; 1967.

[Ma,Mo] Martelli A., Montanari U.: *An Efficient Unification Algorithm*; ACM-TOPLAS, 4(2), pp. 258 – 282, 1982.

[Mi et al.] Mitchell T.M., Keller R.M., Kedar-Ceballi S.T.: *Explanation-Based Generalization : A Unifying View*; Machine Learning 1 47 — 80 ; Kluwer Academic Publishers ; 1986.

[Pa] Patel-Schneider P.F.: *A Decidable First-Order Logic for Knowledge Representation*; Proceedings of IJCAI 9, pp. 455–458, 1985.

[Wa] Waldinger, R. : *Achieving Several Goals Simultaneously*; in Machine Intelligence 6 (eds. E. Elcock and D. Michie), 94 — 136, Ellis Horwood ; 1977.