CADRES:
CAD Design Knowledge Representation System

David W. Franke, David E. Newton, Richard P. Johns

Abstract

CADRES is a design knowledge representation system embedded in the V3 (LISP-based) MCC CAD System. The goals of CADRES are 1) to represent design information in such a way as to facilitate reuse of existing designs and design knowledge and 2) to facilitate reasoning and learning algorithms applied to design information. CADRES provides a framework into which design knowledge is placed, and addresses the following issues: representing design information for reuse; representing design information for reasoning algorithms; representing design generalizations including parameterized design descriptions; searching the space of existing design descriptions; representing constraints; and classifying new design descriptions.

1 Introduction

With increasing complexity of VLSI architectures and implementations, it is advantageous to reuse components of these architectures and implementations in subsequent designs. Reusing such a design description is not limited to the structural description, and can include simulation, timing, layout, and test information. While design descriptions are necessarily captured by all CAD systems, these descriptions need to be classified in a manner which facilitates locating these descriptions in subsequent design work.

CADRES is based on the KL-One system [Brachman, Schmolze 1985]. The KL-One representation language was originally used to support a natural language understanding system. We view the problem of representing design descriptions as similar to that of representing vocabulary, in that we consider design descriptions to be terminological information (definitions of "concepts", in KL-One). Once this network of definitional information has been established, other types of knowledge can be associated with points in the resulting framework.

Given a knowledge base of classified designs and their generalizations, the iterative refinement design model is easily supported. Starting with an abstract description of the design (processor for example) obtained from the knowledge base, search procedures can identify the refinements of the abstraction and its components contained in the knowledge base. These refinements can be used "as is" or they can be the starting point for creating new designs. An alternative source for the abstract design description is one which is entered explicitly by the designer. In this scenario, the existing knowledge base of designs is viewed as a library of components which can be used "as-is", or they can be used as the starting point for new component designs. Finding appropriate designs in this scenario requires sufficient feature specifications and search techniques.

As stated above, these design models can be supported "given a knowledge base of classified designs and their abstractions". Hence, the significant problems in this approach are:

1. Classifying new design descriptions, and
2. Identifying and creating the appropriate design abstractions.

Design abstractions are important in that they are what new designs are classified as "a kind of". Design abstractions are also one source of design plans, which can be reused.

2 Design Information in CADRES

CADRES is based on the KL-One system [Brachman, Schmolze 1985]. The KL-One representation language was originally used to support a natural language understanding system. We view the problem of representing design descriptions as similar to that of representing vocabulary, in that we consider design descriptions to be terminological information (definitions of "concepts", in KL-One). For the most part, we have adopted the terminology and graphic representation of KL-One.

2.1 Concepts

The CADRES (KL-One) object which corresponds to a description is called a *concept.* Concepts are organized into a *structured inheritance network* via a *subsumption* relation. Concept A is said to *subsume* Concept B if all instances of Concept B are also instances of Concept A. There are three important points to make here. First, concepts in CADRES are definitional. Second, we are interested in representing abstract design descriptions, such as Concept A mentioned above. Finally, the subsumption relation also expresses an inheritance relation among concepts. Hence, once Concept A is said to subsume Concept B, Concept B then inherits the description of Concept A.

Consider the (abstract) concept Processor, and the more concrete (but still abstract) concept Processor-With-Cache which is subsumed by the concept Processor, diagrammed in Figure 1. Processor elaborates the definitional aspects common to all processor designs, while Processor-With-Cache elaborates the definitional aspects common to all processor designs which contain one or more caches. Descriptive aspects of Processor are inherited by Processor-With-Cache, and are not repeated in Processor-With-Cache.

The additional concepts Floating-Point-Processor and Floating-Point-Processor-With-Cache can be added, with Floating-Point-Processor subsumed by Processor and Floating-Point-Processor-With-Cache subsumed by Floating-Point-Processor and Processor-With-Cache. Floating-Point-Processor elaborates the definitional aspects of processor designs which provide floating-point arithmetic, while Floating-Point-Processor-With-Cache combines the definitional aspects of Floating-Point-Processor and Processor-With-Cache. The concept Floating-Point-Processor-With-Cache may inherit all its definitional aspects and not have any defined locally.

2.2 Roles

A CADRES concept (design description) elaborates definitional aspect with *roles* and *relationships.* A role defines some aspect of the concept with which it is associated, and records

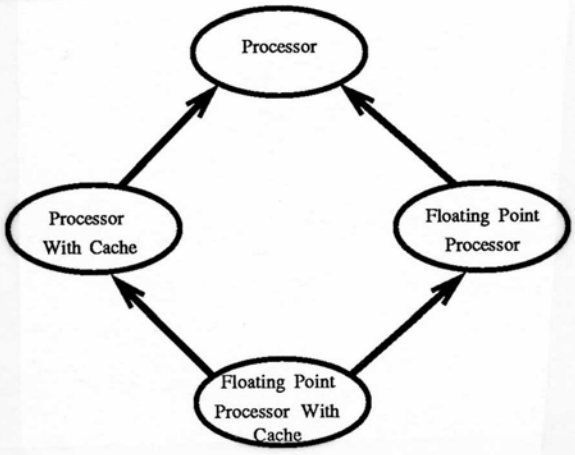

Figure 1: Concepts and Inheritance

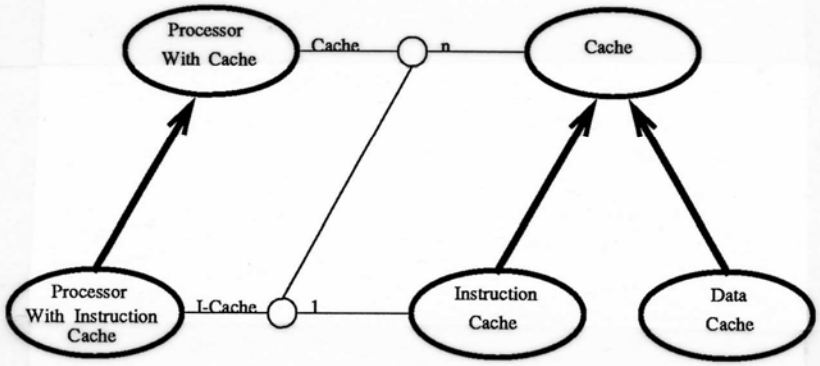

Figure 2: Role

information as to the type and number of objects which can fill this role in an instance of the concept. For example, the concept Processor-With-Cache contains a role (call it cache) which can be filled with one or more objects whose type is the concept Cache (another concept in CADRES), as diagrammed in Figure 2. Roles can also specify their relation to roles in subsuming concepts. For example, the concept Processor-With-Instruction-Cache is subsumed by Processor-With-Cache, and has a role which *restricts* the role cache of Processor-With-Cache by requiring that the objects which fill the role are of type Instruction-Cache, a concept subsumed by the concept Cache. The concept of a processor with multiple caches is specified by *differentiating* the cache role of Processor-With-Cache with several roles, each of which may restrict the type and/or number of potential objects which can fill the role.

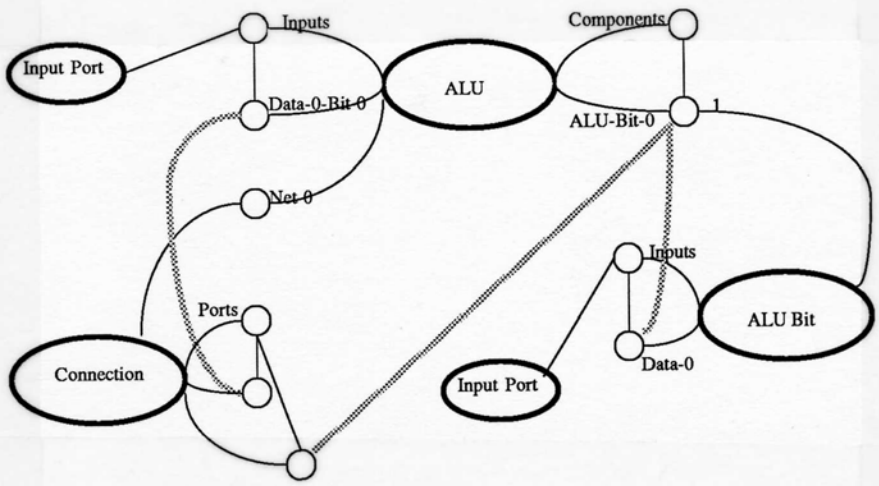

Figure 3: Relationship

2.3 Relationships

A particularly interesting capability of KL-One is the *structural description*, termed relationship in CADRES. (We have adopted the term relationship, as the term structure has a specific connotation in VLSI CAD systems.) A relationship of a concept expresses an interrelation between roles of that concept. We have found several applications of this idea in representing design knowledge, and we believe that there will be more. First, relationships are used to express interconnect between ports in a design. In a design description of an ALU, the net connecting port Data-0-Bit-0 with port Data-0 of component ALU-Bit-0 is expressed as a relationship of ALU which *coreferences* roles Data-0-Bit-0 and (Data-0 ALU-Bit-0), as diagrammed in Figure 3. The expression (Data-0 ALU-Bit-0) is called a *role chain*, and is the mechanism for accessing roles of a potential role filler.

The CADRES relationship has also been employed to represent constraints between roles. In addition to specifying the roles which are involved in the relationship, the relationship itself is typed as to the relation being applied. In the case of nets, the relationship is a connection. In the case of constraints, the relation can be one of many types of constraints available in the system. For example, a constraint between the delays of two components of a design can specify that the delays be equal, or that one must be greater than the other.

Another type of description in which relationships play a part are parameterized design descriptions. Parameterization can be applied to the number of occurrences of a role filler, or to the type of a role filler. Currently, we have implemented parameterized descriptions only with respect to the number of occurrences of a role filler. Consider constructing the concept N-Input-Nor. First, a role defining a parameter (call it number-of-inputs) is added, with the potential role filler restricted to be an integer greater than 1. Next, the role defining the input ports of an N-Input-Nor is defined, with the role fillers restricted to be of type Input-Port and with the number of role fillers restricted to be number-of-inputs. When an instance of this concept is created, the "parameter" role number-of-inputs must be filled with an integer.

When a concept of this sort is being instantiated as a role filler of another concept, the parameter role filler can be obtained from the referencing concept. For example, the concept

Joe's-Design has a role whose filler is restricted to be an N-Input-Nor, a role which is the parameter specifying the bit width of Joe's-Design, and a relationship which specifies how this parameter is related to the parameter role of N-Input-Nor. This "parameter binding" relationship coreferences the parameter role of Joe's-Design and the role number-of-inputs of concept N-Input-Nor, requiring that they be equal. When Joe's-Design is instantiated, an integer value is required for the parameter role of Joe's-Design, and this value is also associated with the number-of-inputs role of the occurrence of N-Input-Nor.

2.4 Rationale

The benefits we believe are gained in expressing design descriptions in CADRES are discussed in some detail below, and are summarized here. First, CADRES provides the mechanisms for representing design abstractions or generalizations. Second, CADRES provides the structure for inheritance, thereby allowing data to be associated with design abstractions and inherited by those design descriptions subsequently classified "under" that abstraction (i.e. the abstraction subsumes the newly classified concept). Third, the uniform representation of design information allows algorithms for classification, generalization, search, knowledge application, and learning to be concerned with a few, conceptually simple, data structures. The real uniformity requirement is in the kind of information required for these algorithms, and hence, this single representation paradigm has been adopted.

3 Design Flow

The manner in which CADRES supports a design reuse strategy is best viewed with respect to the abstraction refinement design model. As stated above, an abstract design description can be retrieved from the knowledge base of designs, or can be entered explicitly by the designer. Once an abstract description is specified, the designer copies and modifies the description in one of the following ways:

- Add more information about the design (select the technology of interest)
- Add to a particular view of the description (structural, dataflow, control flow)
- Add detail to some component of a particular view of the description (select a more specific type for some structural element)

To support adding more information about the design, the representation of the design abstraction in CADRES records the known attributes and aspects of designs of the type that is being done. For example, CADRES records the fact that process technology is an attribute of designs. Once this attribute has been specified to some level of detail, say CMOS, the design abstraction can be classified as a kind of CMOS design. This new classification will then add additional attributes and aspects to the design abstraction, since the design abstraction is now a kind of CMOS design and inherits the attributes and aspects of CMOS designs. The attributes of a design are not limited to those recorded in CADRES, and new attributes can be specified. This new information is then recorded in the CADRES representation of the design.

To support adding information to a particular view of the design description, the knowledge base records information about what views are available and how they are associated. Also, the

CADRES framework provides the structure into which other types of knowledge (rules, plans, etc.) can be placed.

4 Generalizations

Design generalizations are important in CADRES because they are the concepts which new designs are classified "as a kind of". Also, they are a source of abstract designs which can be subsequently refined by the designer. The type restriction and number restriction features of concept roles in CADRES provide a convenient mechanism for specifying generalizations.

The most direct method of capturing design generalizations is to enter them directly via editors. A schematic editing tool can be used to describe a block with substructure, where the substructure components are occurrences of abstract designs known in the knowledge base. For example, to enter the abstract description of a Floating-Point Processor, a block is created with the structural components of Control, Floating-Point ALU, and Memory. These structural components are themselves abstractions, presumably with one or more refinements in the knowledge base. The abstract interface of Floating-Point Processor, namely data input, data output, control input, and control output are also specified. Note that at this level of abstraction, details such as the width (number of bits) of these data and control paths has not been specified. The interconnect among these components and between the components and the interface can also be specified, again with details such as data path width and specific control signals unspecified.

Ideally, these abstractions should be generated by the system itself, either upon demand or unsolicited. Identifying and creating abstractions or clusterings is an area of interest in the field of machine learning [Mitchell 1982], [Fisher 1985], [Lebowitz 1986], [Michalski, Stepp 1986]. While a specific generalization algorithm has not been selected or developed, some types of generalizations which can be made from design descriptions have been identified and are described here with some examples.

Removing definitional aspects: Generalizations can be created by removing specific interconnect, specific ports, and/or specific components. For example, the generalization of a Flip-Flop interface could be generated by removing the specific input ports of known flip-flops such as J-K or S-R.

Generalizing types: Generalizations can be created by abstracting the type of specific aspects of a design description, such as the structural aspects of a design. For example, the generalization Processor-With-Cache could be created from the definitions of different processors which use different types of cache. The resulting abstraction has a structural component whose type is the generalization of the specific caches used in the processors of which this is an abstraction.

Parameterization: Generalizations can be created by recognizing that aspects of design descriptions can be parameterized, both in the number of occurrences of the aspect, and in the type of the aspect. For example, the number of data bits of a Boolean logic gate or of an ALU can be parameterized. The specific type of a structural aspect such as a component or port can be specified as a parameter. Parameterized descriptions not only capture the abstraction, but they also describe how the parameterized aspect might be refined.

The traditional problems of machine learning, namely *When and what to generalize*, can overwhelm the problem of simply representing the abstractions. Our strategy is to take cues

from the designer and design activity to approach this problem. In addition to the CADRES classification of a design, an open classification or indexing facility is provided. This facility allows designers to arbitrarily specify groupings of objects which they have entered or found in the knowledge base, irrespective of their strict CADRES classifications. From the system point of view, these are object groupings created by experts, and are interesting for one reason or another. The designer or CAD System manager could explicitly request that a generalization be created from an index group.

We believe that other information explicitly entered during design or implicit in design activity can assist in addressing the *When and what to generalize?* problem, and this is one area of our current research.

5 Classification

Classification of a concept in CADRES is the problem of placing a new concept in the structured inheritance network. Ideally, when a concept is classified, all possible subsumption relationships are identified and added. Classifying a new design description is limited by the features of the description and the ability of the representation language to express these features. Features such as port type or signal rise time are straightforward to represent, while general representations of function are not. [Brachman, Levesque 1984] and [Patel-Schneider 1986] discuss the computational complexity of classification.

The appropriate classification of new designs is required to support knowledge base search for reusable designs and design knowledge. Classification of concepts in CADRES is one mechanism for organizing or grouping designs. Other design grouping mechanisms are not constrained to the strict subsumption sematics of CADRES, and allow designers to create arbitrary groupings of design objects for their own purposes. Only classification in CADRES is discussed here.

CADRES classification, as in KL-One classification (see [Brachman, Schmolze 1985], [Schmolze, Israel 1983], and [Schmolze, Lipkis 1983]) is concerned with role restriction, both in the type of the potential role filler and the number of occurrences of the role filler. Our approach to providing CADRES classification has been to first provide support for manual classification via an algorithm which can answer the question "Can Concept A be classified under Concept B?". This question is initiated by the designer or CAD database administrator while attempting to integrate (classify) new designs into the existing knowledge base. The next step will be to develop search techniques for identifying potential subsumption relationships. These two capabilities can then be combined to provide some degree of automatic classification. Generalization algorithms must also be developed to identify and create design abstractions under which new designs can be classified.

6 Search

Finding design knowledge for reuse is the primary objective of search. In attempting to find reusable design knowledge, the search algorithms look for the piece of design knowledge that best fits the specific requirements of a particular design situation. We have dubbed this type of search "best-fit" search. Best-fit search can be characterized as an exhaustive, attribute pattern-matching search, and can be expensive, depending on the size of the search space.

The perceived accessibility of reuseable design knowledge should be high. A designer must feel that it is more expedient to reuse an existing piece of design knowledge than to recreate it on the fly, even though that particular piece of knowledge is well understood and easily reproducible. For example, designing an adder is probably second nature to most desginers, but it is still more efficent to reuse an existing adder, since other information besides the structure of the adder can also be reused (e.g. timing data, test data, layout). However, if the process by which that best-fit adder is found and reused disrupts the designer's flow by being slow, complicated, or non-intuitive, designers will take the time to reproduce it.

6.1 Search Space

The potential search space encompasses the domain of VLSI design specifications. The topology of this space is dynamic in that there is a continual influx of new design objects. New design objects are either generalizations or specializations of existing design objects. New subsumption relationships augment the inherited attributes of all concepts in the hierarchicaly below the new subsumption link. The influx of both generalizations and specializations provides the potential for "better-fits". Having once found a best-fit piece of design knowledge for a specific design situation does not guarantee that that same piece of design knowledge will best fit the same design situation at a later time. One of the important strengths of a reuseable design knowledge base is that it reaches beyond the comfortable sphere of design knowledge possesed by a single designer into the larger sphere of collective experience.

6.2 Using Abstractions

Since the search space can be very large (in fact, a larger search space implies that more design knowledge is available), performing a best-fit search conjures up the image of the needle in the hay stack. If reuseability is to be effectively employed in the design process, a way must be found to reduce the size of the search space in which best-fit search takes place. In a more general sense, we must find ways to improve navigation in such a large space.

Recall that an abstraction describes those characteristics shared by all refinements of the abstraction. For example, the abstraction for a processor describes those characteristics of a processor that all processors share: memory, a controller, and an alu as shown in Figure 4. Furthermore, any concept in CADRES subsumed by the processor abstraction is semantically understood to specialize the abstraction, ie., a floating-point processor has memory, a controller and a floating-point alu. In general, an abstraction is any concept in CADRES that is specialized by other objects. For example, floating-point processor could be specialized by floating-point-auto-cache processor. One of the ways in which abstraction helps a designer navigate a large search space is by providing direct pointers to components of an abstract design. The abstract view of a processor might appear as a box with input ports, output ports, memory, a controller and an alu.

For the designer who has never designed a processor before, the abstraction shows the designer what a processor is composed of and provides a template for guiding the design process. If the designer has never designed some particular component of a processor, say the controller, and wishes to find out what makes up a controller, he/she may push into the controller abstraction to see its structure. An abstract view of a controller might appear as a box with

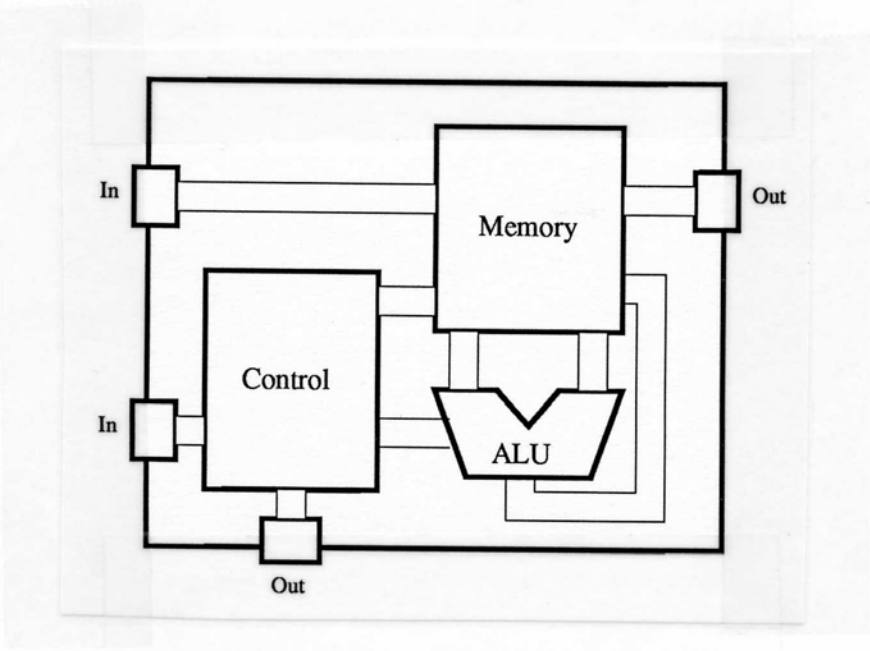

Figure 4: Processor Abstraction

input, output, input logic, output logic and memory. This process of pushing into abstraction is recursive. Eventually the designer must refine each abstraction.

6.3 Refining Abstraction

Refining abstraction is the process of finding the specialization of the abstraction that best fits the specific requirements of the design. It is not always necessary to push down to the lowest levels of abstraction before refining. It really depends on how well the knowledge base is stocked. Initially, the knowledge base may contain only primitive gate level components such as NOR, or NAND. As the the knowledge base matures, more and more complex objects like ADDERs, COUNTERs, REGISTERs, ALUs, CONTROLLERs will populate the knowledge base. To refine abstraction, we use best-fit search.

Best-fit search is an expensive search even for relatively small search spaces like an abstraction subtree. The nature of a best-fit search suggests that the only way to be sure that the best-fit object has actually been found is to look at every object in the search space. The algorithm we use however takes advantage of the fact that the search space is organized hierarchically. We view an abstraction subtree as a collection of subtrees and so on. As such, by incrementally describing the attributes of the best-fit, we hone away those subtrees beneath the abstraction subtree that hold no promise. We have dubbed this algorithm "honing".

Honing is performed by the designer, who interactively adds attribute patterns to a data structure called a "honing-set". A honing-set initially serves as a description of the best-fit. Objects in the search space are matched against the honing-set. Each time an attribute pattern is added to a honing-set, some of the potential best-fit candidates are honed away. In association with each pattern in the honing-set we record the effect each pattern has on the search. This record is demographical in nature because it provides valuable information about the abstraction subtree population. The honing process continues until the best-fit is found. Once the best-fit is found, it is returned to the design tool that originated the search.

In summary, the subsumption hierarchy of CADRES organizes design descriptions in a manner which supports searching and abstraction refinement.

7 Summary

Our work on placing design descriptions in CADRES has directed our implementation, and we have found the features of KL-One to be suitable for representing design information. With the exception of parameterized descriptions, the representational capabilities discussed here have been implemented. Parameterization of the number of occurrences of some aspect has been implemented, while parameterization based on type is in work. Parameterization of conditional aspects (as opposed to iterative, or one or more occurrences) is also under study. However, conditional aspects are similar to cancellation of properties, which is not allowed in KL-One subsumption.

Classification capability is currently limited to manual classification, with a supporting algorithm which can handle questions of the the form "Can Concept B be classification under Concept A?". Classification candidate search techniques are under investigation. The initial implementation of search has been completed, and its capabilities and performance are being researched.

The V3 MCC CAD System has been implemented on LISP workstations in Common-LISP. The CADRES representational capabilities are implemented as described above, and have been integrated into the V3 System. This integration includes saving CADRES objects (concepts) in the storage system which supports the V3 System, and capture of design descriptions entered via 1) a schematic editing tool, 2) a layout editing tool, and 3) VHDL (VHSIC Hardware Description Language) source. An interface for browsing and editing the CADRES network has also been implemented, and currently provides the interface to existing design information.

8 Acknowledgements

The authors would like to acknowledge the assistance of Desmond D'Souza, Raymond P. Voith, Jasbir Brar, and David Burgess in the CADRES implementation and in gaining insights into the issues pertaining to design knowledge representation.

References

[Brachman, Levesque 1984] Ronald J. Brachman, Hector J. Levesque, "The Tractability of Subsumption in Frame-Based Description Languages", in *Proceedings on the Fourth National Conference on Artificial Intelligence,* Austin, August 1984, pp. 34-37.

[Brachman, Schmolze 1985] Ronald J. Brachman, James G. Schmolze, "An Overview of the KL-ONE Knowledge Representation System", *Cognitive Science*,**9**, 1985, pp. 171-216.

[Fisher 1985] Douglas Fisher, "A Hierarchical Conceptual Clustering Algorithm", Technical Report 85-21, University of California at Irvine, 1985

[Lebowitz 1986] Michael Lebowitz, "Concept Learning in a Rich Input Domain: Generalization-Based Learning", in **Machine Learning**, Volume 2, Michalski, R., Carbonell, J., and Mitchell, T. (eds.), Morgan Kaufmann Publishers, Los Altos, CA 1986.

[Michalski, Stepp 1986] Ryszard S. Michalski, Robert E. Stepp, "Conceptual Clustering: Inventing Goal-Oriented Classifications of Structured Objects", in **Machine Learning**, Volume 2, Michalski, R., Carbonell, J., and Mitchell, T. (eds.), Morgan Kaufmann Publishers, Los Altos, CA 1986

[Mitchell 1982] Tom M. Mitchell, "Generalization as Search", *Artificial Intelligence* **28** (2), March 1982, pp. 203-226

[Patel-Schneider 1986] Peter Patel-Schneider, "A Four-Valued Semantics for Frame-Based Description Languages", in *Proceedings of the Fifth National Conference on Artificial Intelligence*, Philadelphia, August 1986, pp. 344-348.

[Schmolze, Israel 1983] James G. Schmolze, David J. Israel, "KL-ONE: Semantic and Classification", in *Research in Knowledge Representation for Natural Language Understanding*, Technical Report 5421, BBN Laboratories, August 1983, pp. 27-39.

[Schmolze, Lipkis 1983] James G. Schmolze, Thomas A. Lipkis, "Classification in the KL-ONE Knowledge Representation System", in *Proceedings of the Eight International Joint Conference on Artificial Intelligence*, Karlsruhe, F.G.R., 1983, pp. 330-332.