A System Which Uses Examples To Learn VLSI Structure Manipulations

Richard H. Lathrop* and Robert S. Kirk**

* MIT Artificial Intelligence Laboratory, NE43-795
545 Technology Square, Cambridge, MA 02139
** Gould/AMI Semiconductors, Inc.
3800 Homestead Road, Santa Clara, CA 95051

ABSTRACT.

Focusing especially on the later stages of the design task, when a complete (or nearly so) design is being optimized at the structural level prior to final physical layout, we identify some aspects of the VLSI domain which complicate efficient design both for human and machine agents. We describe a simple but useful idea and a prototype implemented system which partially addresses these problems. Examples are conveyed graphically from an existing design. The system automatically learns a *design precedent* enabling it to infer local hierarchy corresponding to the example in new designs. The teacher may also substitute alternative actions, described to the system in its native Y hardware description language, which the system remembers and can also apply later. CONSTELLATION can infer local hierarchy; undo and rationalize clever local tricks and work-arounds; search for situations in which a specific local optimization can be applied; and modify the circuit as described by the teacher. The system is in experimental use in a production environment.

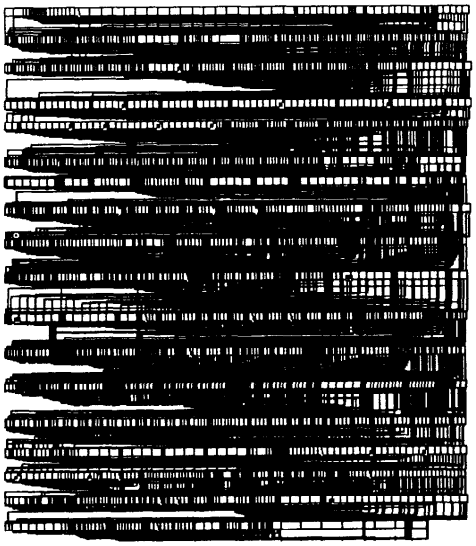

Figure 1. COMET Chip.
Approximately 30,000 transistors comprising about 3,000 cells (at the logic-gate and flip-flop level) and about 3,000 connecting busses. CONSTELLATION often runs at about 1-2 seconds per match-and-replace cycle on this chip.

INTRODUCTION.

Today's complicated VLSI circuits require highly intricate structure. Especially in the later stages of the design process, it is difficult for reasoning systems (human or machine) to verify that the implemented circuit actually reflects the designer's original intent; or to manipulate the structure in a correct, coherent, efficient manner. Two domain attributes exacerbate this:

- The necessarily low level of design descriptions in the intermediate and late design stages obscures the designer's intent with excessive detail.
- The frequent use of clever, situation-specific sub-circuits, obscures the designer's intent with technology-dependent "tricks" and "work-arounds".

Both contribute additional layers to the task of reasoning about (or correctly changing) a design. Even when a circuit was designed using an hierarchical methodology, existing design tools often discard this knowledge soon after the initial design capture. This makes it difficult for subsequent tools or reasoners to exploit regularities or simplifications indicated by the hierarchical organization.

Complexity of circuits in the VLSI domain also impedes the effective application of the designer's expertise. Experienced circuit designers use many techniques to improve the overall quality of a design. In a large and complicated design it is virtually impossible to insure that every optimization a designer may know has been applied everywhere possible. The finished design may be less efficient, even though the designers may know applicable techniques to improve it further, because of the practical difficulty of actually finding a particular place in the design where a particular technique applies.

Knowledge in the VLSI domain takes many forms: global vs. local, structural vs. functional, synthesis vs. analysis, strategy vs. implementation, etc. This paper focuses on local structural knowledge and its application to analysis and refinement of nearly complete designs. We examine a simple but useful idea addressing the sub-tasks of (1) reducing design complexity, by inferring local hierarchy and recognizing clever tricks and work-arounds; and (2) improving overall design quality by performing particular optimizations wherever they apply in a particular design.

The implemented system (CONSTELLATION) described below is a *learning apprentice* in the sense introduced by LEAP [18]: an intelligent tool which is taught by an experienced designer in the course of working on an actual problem. While LEAP addressed generalization in the functional synthesis task beginning with the early stages of a design, CONSTELLATION addresses a different task of structural analysis and refinement operating on a nearly complete design. Our goal is to capture design expertise relevant to this late stage of the design process, and automatically apply it to new designs. We do this by using a construct we call a *design precedent,* loosely defined as, "A situation I have seen before," and, "What to do when I see it again." These pattern/action pairs closely resemble the rules in a traditional rule-based expert system [6, 23].

The major difference between CONSTELLATION and a traditional rule-based system lies in the source and nature of the pattern/action rules. They are captured, not by a Knowledge Engineer, but by the machine under the control of an experienced designer working from an actual design.

The designer indicates a relevant situation by pointing at an example of it, and then describes the appropriate action to take in the Y hardware design language normally used to design circuits. CONSTELLATION's design precedents embody specific local situations seen before and specific actions or tricks applicable there, rather than global heuristic rules. Nonetheless aspects of the knowledge acquisition process described should be partially applicable to traditional rule-based systems.

The system is implemented in LISP on a Symbolics 3600[1] employing an object-oriented message-passing architecture. The largest design that has been run on this system consists of about 30,000 transistors comprising approximately 3,000 primitive cells (at the level of logic gates and flip-flops) connected by approximately 3,000 primitive busses (figure 1). On this chip the system often takes about 1-2 seconds per match-and-replace cycle, rendering it suitable for interactive use.

DATABASE MODEL.

Our database model is based on the Y-diagram proposed by Gajski and Kuhn [9, 11]. The original Y-Database represented the Functional, Structural and Physical aspects of a design as the three arms of a Y. The Functional level describes the behavior and algorithms embodied, and the Physical level describes the geometry and actual layout. The Structural level, which concerns this paper, describes the component modules and their connections. Each **module** (block) has a few named communication **ports**, connected by **busses** (nets) to the ports of other modules. Recursively, each module is internally composed of submodules whose ports are connected by busses, and in this way design hierarchy is realized. We will typically be concerned with a netlist, a list of the modules at the black-box level and of which ports to connect together.

The Star design system coordinates several related design tools. It allows user interaction through both a generic graphical editor and the textual Y hardware description language. Objects in the database are automatically invertible into the Y language, and satisfy a meta-circularity condition that no information be lost or gained in going from internal database representation to text and back. The Y language is embedded in LISP, and Y and LISP statements may be arbitrarily intermixed. The concept is similar to DPL (Design Procedure Language) [2] except that DPL is restricted to physical representations (compare Slices[22]).

[1] A trademark of Symbolics, Inc.

KNOWLEDGE ACQUISITION ENVIRONMENT.

In any system which proceeds by large amounts of domain-specific knowledge, the knowledge acquisition bottleneck must be addressed. In rule-based expert systems this is the difficulty of extracting knowledge from experts, and work on this problem includes [7, 21]. In VLSI it is reflected in the difficulty of accumulating design libraries, in the emphasis on and difficulty of capturing the designer's intent, and so forth. Any system (such as ours) which is based on large amounts of knowledge acquired from human experts must make this process as painless as possible.

The intent of our interface is to have the designer examine an existing design and point to an example where the precedent should apply. The system will produce a precedent which automatically replaces this with a fragment of local hierarchy when the situation is encountered in the future. This default action may be changed by substituting an alternative action, usually described by the designer in the system's native Y hardware description language.

A SESSION WITH CONSTELLATION.

The environment is best described by walking through a session with the system. First an existing design is selected and displayed on the graphics window. Once the design schematic is on the screen, the designer uses the mouse to select interesting components as in figure 2. When all the desired components have been selected the designer selects the Precedent pop-up menu and chooses Make-precedent.

The system then automatically performs the following steps:

1. Isolates and verifies the sub-circuit represented by the selected modules.
2. Inverts the isolated sub-circuit into a Y-language Module Definition (figure 3a).
3. Sets up a SIMMER functional simulation environment and test pattern.
4. Creates a precedent pattern/action rule definition (figure 3b) consisting of:
   (a) a pattern which will trigger the precedent rule when subsequently recognized, and
   (b) an action clause which replaces the recognized pattern with the higher-level module defined in step 2 (a fragment of low-level hierarchy).

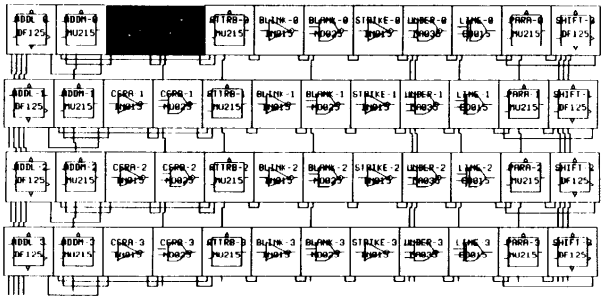

Figure 2. Example datapath circuit.
Two cells (an INVERTER and an NOR gate) have been selected to form a design precedent for the system to record.

```
(DEFMODULE INVNOR 0
  (MODULE 'CSRB-0 'N0025 0)
  (MODULE 'CSRA-0 'IN015 0)
  (PORT 'CB 'BIT 0)
  (PORT 'D2-0 'BIT 0)
  (PORT 'DA-0 'BIT 0)
  (PORT 'DEE-0 'BIT 0)
  (BUS 'CB 'BIT 0)
  (BUS 'D2-0 'BIT 0)
  (BUS 'DA-0 'BIT 0)
  (BUS 'DEE-0 'BIT 0)
  (CONNECT (>> 'CSRB-0.N0025.MODULE 'B.BIT.PORT)
           (>> 'CB.BIT.BUS))
  (CONNECT (>> 'CSRB-0.N0025.MODULE 'A.BIT.PORT)
           (>> 'D2-0.BIT.BUS))
  (CONNECT (>> 'CSRA-0.IN015.MODULE 'Q.BIT.PORT)
           (>> 'D2-0.BIT.BUS))
  (CONNECT (>> 'CSRB-0.N0025.MODULE 'Q.BIT.PORT)
           (>> 'DA-0.BIT.BUS))
  (CONNECT (>> 'CSRA-0.IN015.MODULE 'A.BIT.PORT)
           (>> 'DEE-0.BIT.BUS))
  (CONNECT (>> 'CB.BIT.PORT) (>> 'CB.BIT.BUS))
  (CONNECT (>> 'D2-0.BIT.PORT) (>> 'D2-0.BIT.BUS))
  (CONNECT (>> 'DA-0.BIT.PORT) (>> 'DA-0.BIT.BUS))
  (CONNECT (>> 'DEE-0.BIT.PORT)
           (>> 'DEE-0.BIT.BUS)))
```

Figure 3a. The Y-Language Definition Constructed.

At some later point the designer or a design supervisor system may call up a new design, retrieve previously constructed precedents, and apply them to the new design. CONSTELLATION will search the design for matches to the precedent patterns. When a match is found the action is performed, modifying the circuit as specified. Figure 4 shows the circuit of figure 2, after the precedent of figure 3b has been applied. The design has been simplified by replacing the two-component INVERTER/NOR combination with a fragment of low-level hierarchy, INVNOR, representing a NOR gate with one input inverted. Note that a series of components have disappeared. They have been spliced out and replaced by the new components, which have not yet been redisplayed.

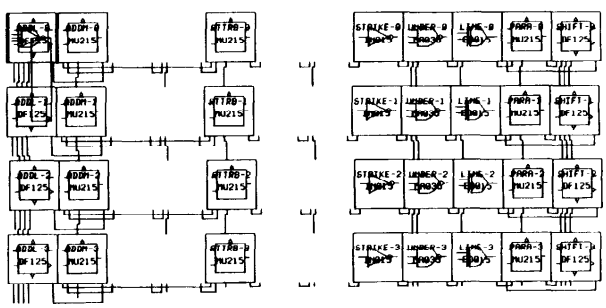

Figure 4. Datapath After Precedent Application. The precedent defined in figure 3b has been applied and the module definition of figure 3a has been spliced in (but not yet redisplayed).

```
(DEFPRECEDENT INVNOR-PRECEDENT
      "A NOR gate with one input inverted."
  :Y-PATTERN
  ((MODULE 'CSRB-0 'N0025 0)
   (MODULE 'CSRA-0 'IN015 0)
   (BUS 'CB 'BIT 0)
   (BUS 'D2-0 'BIT 0)
   (BUS 'DA-0 'BIT 0)
   (BUS 'DEE-0 'BIT 0)
   (CONNECT (>> 'CSRB-0.N0025.MODULE 'B.BIT.PORT)
            (>> 'CB.BIT.BUS))
   (CONNECT (>> 'CSRB-0.N0025.MODULE 'A.BIT.PORT)
            (>> 'D2-0.BIT.BUS))
   (CONNECT (>> 'CSRA-0.IN015.MODULE 'Q.BIT.PORT)
            (>> 'D2-0.BIT.BUS))
   (CONNECT (>> 'CSRB-0.N0025.MODULE 'Q.BIT.PORT)
            (>> 'DA-0.BIT.BUS))
   (CONNECT (>> 'CSRA-0.IN015.MODULE 'A.BIT.PORT)
            (>> 'DEE-0.BIT.BUS)))
  :Y-ACTION
  ((MODULE (UNIQUE-NAME 'INVNOR) 'INVNOR 0)
   (CONNECT (*>> 'CB.BIT.BUS)
            (>> (UNIQUE-NAME 'INVNOR)
                'CB.BIT.PORT))
   (CONNECT (*>> 'D2-0.BIT.BUS)
            (>> (UNIQUE-NAME 'INVNOR)
                'D2-0.BIT.PORT))
   (CONNECT (*>> 'DA-0.BIT.BUS)
            (>> (UNIQUE-NAME 'INVNOR)
                'DA-0.BIT.PORT))
   (CONNECT (*>> 'DEE-0.BIT.BUS)
            (>> (UNIQUE-NAME 'INVNOR)
                'DEE-0.BIT.PORT))
   (ZAP (*>> 'CSRB-0.N0025.MODULE))
   (ZAP (*>> 'CSRA-0.IN015.MODULE))))
```

Figure 3b. The Precedent Definition Constructed.

Reviewing what has happened, the designer explicitly pointed out the precedent with the mouse. The system then automatically isolated the sub-circuit, inverted it into Y-language statements, produced a module definition for the Y design library, constructed a simulation environment, and built a precedent for the precedent library. Subsequently applying the precedent to a design causes CONSTELLATION to search the design for sub-circuits which match the precedent pattern. When these are found the precedent action is executed. The default action created by the system is to infer a new layer of low-level hierarchy in the design, corresponding to the module definition produced when the precedent was created. Figure 4 shows figure 2 after the precedent described in figure 3b has been applied and the low-level hierarchy fragment defined in figure 3a has been inserted (but before it has been redisplayed).

In many circumstances, some more complicated action on the design may be indicated — perhaps recognizing and undoing an complicated and intent-obscuring situation-specific implementation, or applying an optimizing technique to improve overall design quality. In these cases, the designer may describe to the system the appropriate action to perform on recognizing the pattern. These would normally be the same Y-language statements that the designer would encode to effect the change by manually editing the design definition text. By manually editing the precedent definition text instead, the actions can be remembered and applied to other designs under the control of the designer or a design supervisor system.

THE CONSTELLATION SYSTEM.

This section will briefly describe the major components of CONSTELLATION. These include the Database (already described), the Isolator, the Inverter, the Precedent Builder, the Simulation Model Builder, the Matcher, and the Action Evaluator.

Once a precedent has been indicated from within the graphical interface by selecting a group of modules, the Isolator is responsible for figuring out its boundaries and surgically removing it from the surrounding data structures. It does this by following each bus connected to each port of each selected module. Purely internal busses connect only to other selected modules, and are entirely within the precedent boundaries. Otherwise a port is constructed which divides the internal and external parts of the bus (connecting respectively selected and unselected modules). The port type (INPUT, OUTPUT, I/O, etc.) is inherited from the internal port types to which it connects.

The excised precedent is then passed to the Inverter, which generates the design library description of the composite block, figure 3a. The Precedent Builder modifies this suitably to serve as the precedent pattern, creating a default action to produce the precedent library entry, figure 3b. The default action is to disconnect the recognized modules from the circuit where they occur, to instantiate in their place the design library entry from the Inverter, and to connect the new ports into the circuit. In this way a fragment of hierarchy can be automatically reconstructed. Arbitrary Y or LISP statements may be manually substituted for the system-generated default action.

The isolated precedent data structure, together with external ports, is passed to the Simulation Model Builder. This module retrieves the SIMMER [14] functional simulation models of the component blocks, and constructs a **structural block** functional simulation library model.

When the precedent is actually applied, the Matcher enumerates the instances in the new design which match the precedent pattern. A standard depth-first graph-matching search is used. For efficiency, only those neighborhoods which potentially contain part of the precedent are searched. When an instance of the pattern is found, matching data structures from the database are bound and passed to the Action Evaluator. The Action Evaluator executes the action part of the precedent (which may be statements in the database language, or arbitrary LISP code).

APPLICATIONS.

CONSTELLATION is currently in use on an experimental basis in a production environment. Gould/AMI operates a silicon foundry and often takes in designs from customers for fabrication. These frequently arrive in a netlist format with no hierarchy information. There are three general areas where we have applied the tool. First is inferring hierarchy in order to reduce design complexity. Second is circuit criticism aimed at improving the overall quality of the design. The last addresses silicon compilation.

Inferring Hierarchy.

Highly structured and ordered systems are much easier to deal with than unstructured, complicated ones. Design software could do a more efficient job in terms of chip performance and area efficiency if additional hierarchy information could be obtained. One application of the precedent-based reasoning system is to infer hierarchy from non-hierarchical structures. By applying different rule sets, different hierarchies can be extracted.

We found that different foundry customers were combining the standard cell set in similar ways to create higher-level functions. In some cases it became obvious that special optimized cells should be created to reduce chip area. On the large 3,000 cell design of figure 1, we discovered a 62 cell pattern which was part of a larger 500 cell regular structure which could be laid out much more efficiently by taking advantage of the regular structure.

Circuit Criticism.

The basic pattern recognition capabilities of the precedent-based reasoning system lends itself to design criticism. By pointing out examples of bad circuit design practice, a library of criticism precedents can be developed. Presently the knowledge acquisition environment builds only the default action form directly, but the suggestion-reporting action can easily be substituted manually.

In practice it is useful to combine some hierarchy inference precedents with the circuit criticism precedents. The circuit is modified considerably by the hierarchy inference precedents to bring out major design abstractions explicitly and to alter logic structures which obscure the functions of the circuit. Once the circuit is reorganized, the criticism precedents are applied. We expect that, as confidence in the correctness of the precedent-based suggestions increases in this sensitive and critical production environment, more of the actual circuit modification will be handled by this tool as has been demonstrated in the laboratory.

Silicon Compiler Optimization.

In the future we plan to develop a connection to a Function-to-Structure silicon compiler. In anticipation of this we have explored the use of CONSTELLATION as an optimizing post-processor. Function-to-Structure silicon compilers typically deal with high-level architectural issues and hence produce netlists which are optimized at the global (system) level. There are usually additional local (logic-level) optimizations which can be performed on the final netlist. (In fact, these optimizations can be performed on any netlist, regardless of source.) A group of precedents could be accrued which were customized to the idiosyncrasies of a particular silicon compiler, by inspecting its output designs and indicating where optimizations should be made.

RELATION TO OTHER WORK.

The basic idea of precedent-based reasoning arose in machine-learning research, from intuitions that a situation seen before should be of assistance in understanding a new situation. Winston et al. [25] attempted to infer potential functions of a novel device by noticing similarities between its structure and the structure of familiar devices whose function is known. The ideas of Minsky[17] on global computation through the interaction of numerous small, local pieces of knowledge have also strongly influenced the conceptual design. Precedents as we use them are accordingly less complex than in [25], and so each precedent can be considered a small "expert" on its own small patch of structure.

In the VLSI domain Ressler [19] in analog (op amp) design, and Hall [10] in digital design, have applied the idea of a "design grammar" to the task of design synthesis. Brotsky [4] has implemented a fast parsing algorithm for some graph grammars. The LHS and RHS of these grammar rules correspond to the pattern and action parts of our precedents. Cliches were used as abstracted templates of commonly-used software-engineering strategies in the Programmer's Apprentice [20, 24] for automatic programming. Kramer [13] has investigated the application of cliches to the control of reasoning in the VLSI design synthesis task.

Attempts to automatically group elements of structure together have been reported for a schematic diagram graphical display [1] and a test pattern generator [8]. Although DPL was purely a physical level language, elsewhere Sussman's concept of Slices [22] (the re-expression of part of a design in a different representation) explores a general abstraction mechanism. The LEAP learning apprentice program [18] proposed an intelligent tool which the expert can uses in the normal course of solving a problem.

The first rule-based system ever constructed (DENDRAL [5, 16]) used (chemical) structure graphs in the if- and then-parts of its rules. Manually editing the precedent is similar to the "copy&edit" mentioned by Lenat [15] for rule creation, except that the total universe of potential objects to copy is all of VLSI design space (huge) rather than existing-rule space (a few hundred). Rule-based expert systems have already proven useful in VLSI (e.g., [3, 12]). We anticipate that our precedent library could be incorporated into a pattern-directed inference control mechanism, providing some of the low-level rule knowledge required.

CONCLUSION.

Precedent-based reasoning in CONSTELLATION provides a tool for capturing procedural structural knowledge about a design. The knowledge acquisition environment permits the designer to simply point out a pattern and specify what is to be done when the pattern is found in a circuit. This provides an easy way to capture and routinely apply simple design transformations in common design situations. Other and more powerful reasoners will be necessary to guide major design decisions, and to reason about special cases for which no precedent is known. The underlying database, Y language, and computation environment serve to make the precedent pattern/action rule mechanism capable of supporting a diverse range of applications. Future developments will focus on generalizing the precedent pattern/action processor, and use of information on the functional and physical arms of the Y diagram.

ACKNOWLEDGMENTS.

The authors would like to thank Mark Alexander, Gavan Duffy, and John Mallery for their contributions to this project. Comments from Kelly Cameron, Richard Doyle, Bob Hall, Walter Hamscher, David Kirsh, Jintae Lee, Doug Marquardt, Bruce Richman, Brian Williams, and Patrick Winston were also very valuable. Personal support for the first author was furnished by an IBM Graduate Fellowship, and during the early stages of this research was furnished by an NSF Graduate Fellowship and by a research/teaching assistantship from MIT. This paper describes research performed jointly at the MIT Artificial Intelligence Laboratory, and at the AMI Cad Research Laboratory. Support for the MIT Artificial Intelligence Laboratory's research is provided in part by the Advanced Research Projects Agency under Office of Naval Research contract N00014-80-C-0505.

REFERENCES.

[1] Arya, Anjali, et. al; "Automatic Generation of Digital System Schematic Diagrams", 22nd IEEE Design Automation Conference (DAC'85), Las Vegas, Nev., June 23-26 1985, paper 24.4, pp. 388-395.

[2] Batali, John; Hartheimer, Anne; "The Design Procedure Language Manual", M.I.T. Artificial Intelligence Laboratory Memo 598, Sept. 1980.

[3] Brown, Harold; Tong, Christopher; Foyster, Gordon; "Palladio: An Exploratory Environment For Circuit Design", *Computer,* Dec. 1983, pp. 41-56.

[4] Brotsky, Daniel C.; "An Algorithm for Parsing Flow Graphs", M.I.T. Artificial Intelligence Laboratory Technical Report 704, March 1984.

[5] Buchs, A., et al.; "Applications of Artificial Intelligence for Chemical Inference VI. Approach to a General Method of Interpreting Low Resolution Mass Spectra With a Computer", *Helvetica Chemica Acta,* 1970, 53:1394.

[6] Davis, R. and Lenat, D.; *Knowledge-Based Systems in Artificial Intelligence,* McGraw-Hill (New York), 1982.

[7] Davis, R.; "Knowledge Acquisition in Rule-Based Systems: Knowledge About Representations as a Basis for System Construction and Maintenance", in *Pattern-Directed Inference Systems,* D. Waterman and F. Hayes-Roth (ed.), Academic Press (New York), 1978, pp. 99-134.

[8] Delorme, C., et al.; "A Functional Partitioning Expert System for Test Sequences Generation", 22nd IEEE Design Automation Conference (DAC'85), Las Vegas, Nev., June 23-26 1985, paper 47.5, pp. 820-824.

[9] Gajski, Daniel D.; Kuhn, Robert H.; "Guest Editor's Introduction: New VLSI Tools", *Computer,* Dec. 1983, pp. 11-14.

[10] Hall, Robert; *On Using Analogy to Learn Design Grammar Rules,* S.M. thesis, Massachusetts Institute of Technology, Dec. 1985.

[11] Healey, Steven T.; Gajski, Daniel D.; "Decomposition of Logic Networks Into Silicon", 22nd IEEE Design Automation Conference (DAC'85), Las Vegas, Nev., 23-26 June 1985, paper 13.1, pp. 162-168.

[13] Kramer, Glenn A.; "Representing and Reasoning about Designs", unpublished manuscript.

[12] Kowalski, T. J.; Thomas, D. E.; "The VLSI Design Automation Assistant: What's in a Knowledge Base", 22nd IEEE Design Automation Conference (DAC'85), Las Vegas, Nev., 23-26 June 1985, paper 18.1, pp. 252-258.

[14] Lathrop, Richard H.; Kirk, Robert S.; "An Extensible Object-Oriented Mixed-Mode Functional Simulation System", 22nd IEEE Design Automation Conference (DAC'85), Las Vegas, Nev., 23-26 June 1985, paper 39.2, pp. 630-636.

[15] Lenat, D.; Prakash, M.; Shepherd, M.; "CYC: Using Common Sense Knowledge to Overcome Brittleness and Knowledge Acquisition Bottlenecks," *AI Magazine,* vol. 6, no. 4, pp. 65-85.

[16] Lindsay, R., et al.; *DENDRAL,* McGraw-Hill (New York), 1980.

[17] Minsky, Marvin; *Society of Mind,* Simon and Schuster (New York), to appear 1986.

[18] Mitchell, Tom M.; Mahadevan, Sridhar; Steinberg, Louis I.; "LEAP: A Learning Apprentice for VLSI Design", Proc. 9th Intl. Joint Conf. on Artificial Intelligence (IJCAI'85), Los Angeles, Ca., 18-23 Aug. 1985, vol. 1, pp. 573-580.

[19] Ressler, Andrew L.; "A Circuit Grammar For Operational Amplifier Design", M.I.T. Artificial Intelligence Laboratory Technical Report 807, Jan. 1984.

[20] Rich, C.; "Inspection Methods in Programming", M.I.T. Artificial Intelligence Laboratory Technical Report 604, June 1981.

[21] Smith, Reid G., et al.; "Representation and Use of Explicit Justifications For Knowledge Base Refinement", Proc. 9th Intl. Joint Conf. on Artificial Intelligence (IJCAI'85), Los Angeles, Ca., 18-23 Aug. 1985, pp. 673-680.

[22] Sussman, Gerald J.; "Slices: at the Boundary Between Analysis and Synthesis", in *Artificial Intelligence and Pattern Recognition in Computer Aided Design,* J.-C. Latombe (ed.), North-Holland Pub. Co. (Amsterdam), 1978, pp. 261-299.

[23] Waterman, D. and Hayes-Roth, F. (ed.); *Pattern-Directed Inference Systems,* Academic Press (New York), 1978.

[24] Waters, R. C.; "The Programmer's Apprentice: A Session with KBEmacs", *IEEE Trans. on Software Eng.,* vol. 11, no. 11, pp. 1296-1320.

[25] Winston, Patrick H., et al.; "Learning Physical Descriptions From Functional Definitions, Examples, and Precedents", M.I.T. Artificial Intelligence Laboratory Memo 679, January, 1983.