AI Magazine Volume 3 Number 3 (1982) (© AAAI)

Heuristic Search for New Microcircuit Structures: An Application of Artificial Intelligence

Douglas B. Lenat
Heuristic Programming Project
Department of Computer Science
Stanford University

William R. Sutherland
Sutherland, Sproull, & Associates, Inc

James Gibbons
Stanford Electronics Laboratory
Stanford University

Summary

Eurisko is an AI program that learns by discovery We are applying Eurisko to the task of inventing new kinds of three-dimensional microelectronic devices that can then be fabricated using recently developed laser recrystallization techniques Three experiments have been conducted, and some novel designs and design rules have emerged.

The paradigm for Eurisko's exploration is a loop in which it generates a new device configuration, computes its I/O behavior, tries to "parse" this into a functionality it already knows about and can use, and then evaluates the results In the first experiment, this loop took place at the level of charged carriers moving under the effects of electric fields through abutted regions of doped and undoped semiconductors. Many of the well-known primitive devices were synthesized quickly, such as the MOSFET, Junction Diode, and Bipolar Transistor. This was unsurprising, as they were short sentences in the descriptive language we had defined (a language with verbs like Abut and ApplyEField, and with nouns like nDopedRegion and IntrinsicChannelRegion) Future innovation at this level would require adding a large knowledge base of physical equations, judgmental rules for employing them, and numerical routines for efficiently applying them

Rather than carrying out this expansion, we decided to seek a higher, less numerical level of abstraction at which to design new devices Previously, our program had worked at a purely geometric level, where each region had a precise size, shape, and orientation. Briefly, we considered a purely topological level, where only conduction paths mattered This proved too abstract, as most of the designs that were efficient at that level were unrealizable geometrically. We derived an intermediate level, that of "tiles." The tile model posits that each region (channel, metal, etc) is a cube (or other space-filling element) of three-space of approximately the same size A device is a lattice of tiles in a particular 3-D configuration The philosophy was that if Eurisko could produce an interesting design at that level, Gibbons could find a way to fabricate it using the new high-rise chip techniques.

In our first attempts to work at the tile level, Eurisko carried out systematic exhaustive searches with few useful new designs This gave us an appreciation for the size of the search space. During one week-long run, the program serendipitously synthesized a very compact three-dimensional design for a flip-flop

Our final experiment was also at the tile level, but in this case Eurisko employed a body of heuristics to guide the synthesis of new devices These informal judgmental rules served as both plausible move generators and implausible move eliminators Almost immediately, the "Symmetrize" heuristic produced a very powerful yet simple device, one which simultaneously computes NAND and OR using only two small metal regions, two n-doped regions, two p-doped regions, and one intrinsic channel region. These devices are composable in two ways—

We wish to thank those graduate students who have aided us in the construction of RLL, the language in which Eurisko is written, most notably Greg Harris at CMU and Russ Greiner at Stanford. For advice and ideas during the design and construction of Eurisko, we thank John Seely Brown, Bruce Buchanan, Ed Feigenbaum, Mike Genesereth, and Herb Simon Harold Brown and Mark Stefik and their collaborators paved the way for the encroachment of AI into VLSI design, and we are grateful to them for reports of their efforts. The excellent computational facilities of Xerox PARC's CIS made it possible to build Eurisko quickly and to run it for long periods of time on several machines. A round of thanks to the Interlisp-D group both for their language and for excellent response to suggestions and bugs Theoretical aspects of this research were funded by ONR N00014-80-C-0609.

Relevant Recent Developments

Some important AI advances that underlie our explorations of this interdisciplinary synthesis include

1 Operational AI programs that can reason—and discover new concepts—in specialized domains of knowledge,

2 AI systems that can introspect and monitor their own performance and discover new algorithms and heuristics,

3. Emergence of the AI subfield called knowledge engineering and its accumulation of experience in applying heuristic methods to problem solving, search, and design tasks in several technical domains

Some important microelectronic fabrication advances that provide important assumptions to this and other projects include:

1 Recrystalization of silicon films to make multilayer "high-rise" circuit structures, and structures no longer limited to wafer-sized areas,

2 The use of (nearly-)intrinsic channel material—that is, a region that can support a channel of electrons or holes, depending only on what types of doping its neighboring regions exhibit,

3 Increasing computer control of fabrication equipment—fabrication operations are becoming more precisely defined and thus more faithfully representable inside a knowledge base,

4 The potential to merge CAD and CAM in the electronics domain into a new form of operation where design, fabrication, and testing are intertwined,

5 Doping patterns need not be rectangular, but may be hexagonal or even irregular The limiting factor is no longer "difficulty of fabrication," but rather "complexity of design "

Figure 1 The recent developments upon which this research is founded

they pack into the plane and they stack on top of each other, so the asymptotic number of regions used is reduced to *one* of each type The device is unconventional in that currents of both holes and electrons must coexist (and cross) inside a single thin film layer In April 1982, the design was "proved" by Gibbons by successfully fabricating them in his laboratory These devices now form the primitive building blocks of Eurisko's latest high-rise chip designs

Besides the useful devices, we now have a few useful heuristics for the task of designing three-dimensional VLSI circuits: in every second metal layer, wires should run N-S (and in the other metal layers, E-W); any 3-D folding of a 2-D design should replace (most of) the pairs of gates sharing a common control by single pieces of metal serving simultaneously as gates for regions above and below them; etc Our current research is aimed at getting Eurisko to design three-dimensional microelectronic devices for more complex functions and to discover additional informal rules for designing such circuits

Background: Recent Developments in AI and Microelectronic Fabrication

The late seventies produced technological improvements in both microelectronic fabrication capabilities and in artificial intelligence systems (see Fig. 1). These parallel developments have suggested using AI techniques for designing or discovering new microelectronic circuit structures. This article reports on one such interdisciplinary research effort.

Relevant recent developments in microelectronic circuit fabrication. One central advance in microcircuit fabrication methods that is motivating our efforts is the ability to build three-dimensional structures--so-called "high-rise chips." A thin film of deposited silicon can be melted in place and recrystallized into transistor-grade material. Appropriate related steps of doping, oxide growth, and metal deposition have been shown to produce working devices. Lee et al. (1978) showed that both enhancement and depletion mode devices could be made in recrystallized polysilicon, and Gibbons and Lee (1980) produced a one-gate-wide CMOS inverter in which a single piece of metal served as the gate control for two transistors—one above the metal and one below it.

One begins with a normal two-dimensional MOS chip, adds a new layer of amorphous silicon "frost" on top of it, scans and recrystalizes it with a laser, dopes the new silicon layer, deposits a new gate-oxide and metal pattern on top of that, then deposits a new layer of silicon frost, scans it, etc. In this way, arbitrarily many alternating layers of metal and semiconductor are assembled into a "high-rise chip," much like a layer cake or a piece of plywood. The resulting structure contains many active devices in a very small volume. To date, the number of layers has been small (only a few 3-ply layers), but this is due more to a lack of worthwhile designs to fabricate rather than to technical problems with the process.

An intriguing opportunity is to dispense with the "bottom silicon crystal" entirely—just start with, say, a sheet of glass. In this way, the area of the structure is not limited to single-crystal wafer-sized regions, but could be arbitrarily large (e.g., the size of a CRT display screen, or a dashboard, or a desk-top). While these possibilities are exciting and probably will be important in the future, they would demand

a major laboratory retooling One big advantage of the laser recrystallization process is that many existing commercial VLSI fabrication facilities already have the equipment required for that process and, thus, can produce high-rise chips provided they are at most wafer-sized.

One concern is that of *yield.* Given no dramatic increase in the reliability of fabrication techniques, any enormous structure is bound to contain many faults. Besides the usual design techniques for overcoming this (e.g., redundancy), a new technology that may be important is the ability to dynamically monitor the complex structure as it is being fabricated, layer by layer. That is, one might fabricate "with the power on." If part of one layer has too many faults, the program might alter the design of the rest of the layer at that moment before the computer advances the laser or ion implantation beam If an entire layer is unacceptable, one might even evaporate it and try again. This merging of CAD and CAM might open up a new paradigm for the design and production of microelectronic structures.

The second major innovation we are relying on is the use of intrinsic—or nearly-intrinsic—channel material. This is a region of semiconductor material that is so lightly doped that it can support a channel of electrons or holes, depending only on what types of doping its neighboring regions exhibit. This advance, like the one before it, explodes the size and complexity of the design problem.

As more and more of the fabrication process comes under computer control, it becomes easier—and more worthwhile—to represent fabrication knowledge within a knowledge base. Once the knowledge is accessible to artificial intelligence programs, they can carry out tasks involving planning, dynamic monitoring and replanning, and (as we shall discuss in detail) guided exploration for novel devices, mask configurations, fabrication sequences, and post-production tailoring via recrystallization.

With computers doing much of the design and fabrication, there is less need to make the doping patterns rectangular. For example, if a gate is to join five regions, we can shape it as a hexagon (still leaving one edge of the hexagon free to have metal deposited over it prior to doping, if that is the order of fabrication steps that finally is employed).

All of these notions are exciting. The excitement is dimmed slightly by the question: "What do we use all this for, exactly?" To answer that question, we turn to a very different part of computer science—artificial intelligence.

The limiting step in the VLSI world is not so much fabrication techniques as coping with the combinatorial explosion of the design problem. A vast new and as yet unexplored design space for three-dimensional microcircuit elements is being opened. The size of this search space makes it a promising domain for some kind of AI approach Recent developments in AI suggest trying an automated induction approach, that is, building a program that learns—by discovery—new physical devices, circuits, and more or less informal rules for designing them What are the AI results that suggest this?

Relevant recent developments in artificial intelligence. In the past decade, a style of AI program has emerged known as expert systems. These programs work in real-life technical domains, such as medical diagnosis, mineral or oil exploration, and planning genetics experiments (see, e.g., Feigenbaum, 1977). The human expert works with a computer scientist, known as a *knowledge engineer,* who enters the expert's knowledge into a program. The first body of knowledge extracted from the expert invariably comprises terms, facts, standard procedures, etc.—the kind of knowledge one would read about in journals and textbooks

But programs with only this type of information do not perform well. A cyclic procedure is followed to improve the program—the knowledge engineer has the program try to work on a problem or case, the expert disagrees with its reasoning at some point, and the knowledge engineer forces the expert, at that moment, to introspect on what extra knowledge he or she is bringing to bear. This usually results in the expert explicating a judgmental rule, an informal bit of wisdom we refer to as a *heuristic.* These heuristic rules are rarely discussed verbally. Rather, the apprentice (intern, graduate student, etc.) is expected to induce them from experience by watching a master at work. As more and more heuristics are added to the program, it incrementally approaches the expert in competence at the task.

Most AI texts talk about heuristics as if they were rules that helped you constrain a search by pruning away implausible moves Some of the heuristics that emerge in the knowledge engineering cycle do indeed have this flavor, but many heuristics appear to serve a very different role—they propose plausible moves to try, plausible promising actions to carry out in certain situations In the "implausible constrainer" sense, as you add heuristics you have less and less to consider; in the "plausible suggester" sense, when you add more heuristics you have more to do. This notion of heuristics as plausible move generators is useful when the space to be explored is too immense to even consider having a legal move generator as your primary engine.

The field of knowledge engineering has by now contributed much in the way of representations for knowledge, control structures for managing large collections of heuristic rules, languages that facilitate the construction and debugging of enormous programs, ways of validating the performance of expert systems, etc. One can think of our task—designing three-dimensional microcircuits—as such an expert task It involves a search through an enormous space of possibilities, there is much technical information to represent and use, it is a task of interest and practical utility, and it is a domain where no algorithmic approach is known at present. In short, it satisfies most of the criteria for attack by the knowledge engineering approach.

The nature of the three-dimensional microcircuit design task is one of open-ended exploration in a large space where the goals are criteria such as lower power, higher gain, smaller space, less time, fewer masks, etc. These criteria are not too difficult to test, but they are of little aid in in-

venting novel designs that satisfy them. Although most expert systems have performed tasks that were classificatory (e g , diagnosis), only some have dealt with problems of design, and a couple have tackled problems that involved open-ended concept discovery and exploration (see, e.g., AM, Davis and Lenat, 1981; Browser, Dankel, 1979).

AM was given the definitions of a hundred concepts from finite set theory and a body of two hundred heuristic rules that guided it in forming plausible definitions, gathering data about those new concepts, noticing regularities in the data, forming plausible conjectures thereby, designing and carrying out experiments to test them, and (to close the loop) extracting useful new definitions based on these results.

To get the flavor of AM, consider this mathematics heuristic:

> Given an interesting function $f : A \times A \rightarrow B$,
> It's worthwhile defining and studying $g(x) = f(x, x)$

When f is Multiplication, the derived function g is Squaring; when f is Addition, the heuristic causes us to define and study Doubling. When f is a predicate such as `Greater-Than`, it causes us to notice that a number is never greater than itself. When f is Intersection, it points us to the fact that `Intersect`$(x, x) = x$. The use of this heuristic is not limited to mathematics, of course. We could make f the binary relation `Employed-By`, in which case g defines the predicate `Self-Employed`. Once a new concept has been defined, it is often a relatively straightforward affair to find instances of it and then to look for patterns in that data (see Polya, 1945).

In one hour-long run on a KI 10, the AM program defined two hundred new concepts, of which about half were reasonable, recognizable mathematics objects (including the empty set, natural numbers, and primes), operators (including compose-with-itself, addition, factoring), and conjectures (including de Morgans laws, the unique factorization theorem, and a strange regularity involving numbers with very many divisors).

One difficulty with AM was that, as it began to work in fields further and further removed from set theory, its initial set of heuristics was not adequate to guide it away from implausible concepts (e.g., numbers that are both odd and even) and toward plausible ones. This need for new, domain-specific heuristics for each new field raises a serious problem with our microcircuit design task.

The task of exploring the space of high-rise chip designs is analogous to AM exploring the space of set theory concepts in all major respects save one—there are as yet no human experts in the field. There is no one who knows what the heuristics are. It is easy to find and program hundreds of heuristics for dealing with sets and functions, but it is impossible to do so for high-rise chips. The few people now working in the field are employing analogues of heuristics from two-dimensional VLSI design, which in turn got most of its heuristics from even older technologies where, for example, wires were cheap and small entities The economies, tradeoffs, and opportunities for local optimizations and counterintuitive designs are very different in VLSI design, and even after several years only a partial set of design heuristics has emerged. We expect just as radical a change in design when going from two to three dimensions, and *all* of those new heuristics are waiting to be discovered.

The relevance of AI does not end at this point. We refer to a program that learns new heuristics by discovery, inducing them from experiences it has while exploring. This program, Eurisko, is described at length in Lenat (1982a, 1982b). In brief, its presumption is that just as a body of heuristics was able to guide AM in discovering set theory concepts, a body of heuristics might be able to guide a program in discovering, testing, and modifying new heuristics This might seem to be dangerously circular, but in fact some useful results have been obtained by applying a heuristic *to itself*. Consider the following heuristic:

> IF F is sometimes useful but sometimes just takes up a lot of time,
> THEN try to find some specializations of F.

This rule was relevant and useful sometimes, but sometimes took up a lot of time to apply. Therefore, the heuristic was relevant to itself. Eurisko applied it to itself and produced several new, more specialized heuristics, a useful one of which was:

> IF F is sometimes useful, but *usually* just takes up a lot of time,
> THEN try to find some *extreme* specializations of F.

Eurisko explored several domains, including set theory, number theory, games, biological evolution, and the design of naval fleets. The latter is perhaps of most relevance to our circuit design task Eurisko was given two hundred pages of rules and constraints on designing individual ships, plus a simulator which allowed it to determine which final fleet could beat which other fleets. It then designed fleet after fleet, using its simulator as the "natural selection" mechanism as it "evolved" better and better fleet designs The search space—the number of parameters—was much too large for any sort of systematic (e.g., linear programming) or even monte carlo approach to the problem to succeed

For instance, when one fleet beat another, Eurisko had to analyze the differences between them, which usually meant analyzing the differences between individual ships. Even once a single parameter appeared to be important (e.g , one fleet was more heavily armored), experiments must be done (i.e., new designs made) to investigate the overall benefits of armoring.

After a while, Eurisko also noticed another kind of regularity. For almost each parameter, the optimal value seemed to be almost, but not quite, an extreme value. This was formed into a heuristic rule that enabled Eurisko to very rapidly settle in on a winning fleet design. That new heuristic said

> IF designing either an individual ship or a fleet for Traveller TCS, and a certain parameter is having its value changed,

THEN change it to a nearly—but not quite—extremal value.

The final fleet Eurisko designed had a large number of ships—each was fairly small, each had nearly as many types of weapons as allowed, each was nearly as heavily armored as possible, each was nearly as slow as possible, etc.

Each evening during June 1981, Lenat would start Eurisko running on this task on one or more Xerox 1100 (Dolphin) machines. All night it would try out new designs and occasionally new design rules and run simulated battles to evaluate them. In the morning, a quick glance over its new ideas would be in order to occasionally give an extra reward to one that Eurisko failed to appreciate fully. For example, one morning Lenat noticed a fleet that was decimated, except for a small lifeboat that could not be defeated because it had been designed with incredibly expensive computer controls that enabled it to outmanuever all incoming fire. The significance of this was not fully appreciated by Eurisko, but we made sure to include one such small unhittable craft in the final tournament fleet. On July 4, 1981, that fleet won the national (Traveller TCS Origins) tournament by winning seven consecutive battles. Following the victory of Eurisko's highly unconventional fleet, some of the rules were changed for this year's tournament. In particular, repairing of ships is no longer permitted and this turns out to eliminate the usefulness of the small unhittable ship.

Because the general design heuristics for Traveller TCS are (probably) still valid, even with the changes in the rules, it should take Eurisko much less time to design a good fleet for this year's competition. If it had not abstracted its experiences into heuristics, we would have had to start Eurisko all over again this year, slowly evolving a fleet design. There was a regional tournament in the Bay Area over Washington's birthday in which a dozen rule changes were announced only a couple days before the event. Since the design heuristics were still valid, Eurisko did come up with a good design in two days and its fleet won that tournament. One rule change was that victory is now tied to monetary damage, not ultimate survival Another rule change involved limiting the number of exchanges of fire to 40. Eurisko tried nearly-extreme designs and came up with a ship that in many ways is the opposite of its little lifeboat. This new ship is large, has huge weapons, but no defense whatsoever. Since there are only forty rounds of fire allowed, one builds forty of these ships and puts one up each round. Yes, it gets sunk, but since most of one's money goes for defenses in Traveller TCS, the monetary damage one inflicts on the enemy is great The rule changes had a great impact on the final designs, but little on the heuristics. Because it discovered a body of design rules, Eurisko became (in a small way) an expert at designing Traveller TCS fleets.

Research with Eurisko in various domains has established its ability to learn simple judgmental rules by abstracting them from experience, by modifying existing rules, and occasionally by analogy to existing rules. This ability to discover heuristics is crucial in our high-rise chip design task, where task-specific heuristics probably exist and are necessary for good performance, but are as yet undiscovered by people. Another way of saying this is that we are embarked on the task of creating an "expert system" for a field in which there are as yet no human experts to copy from or learn from and that we propose to try doing this through Eurisko's ability to discover useful heuristic expertise itself.

The Opportunity: Using AI Methods to Search for New Microcircuit Structures

With the above technological capabilities as background, in July 1981 we began to consider how to apply AI capabilities to the synthesis of novel microelectronic structures. Our basic approach has three simple and obvious steps:

1. The program starts with some primitive microcircuit concepts as built-in knowledge along with simple rules and evaluation criteria.
2. Using composition rules, it combines several known entities into a new one. In rare cases, a rule takes a single known entity and mutates it
3. This new structure, rule, operation, etc is then evaluated for interest and either retained or junked as appropriate

The most common instance of action 2 is to take known primitive (or complex) microcircuits and produce new ones. More rarely, new heuristic rules are produced. Even more rarely, new evaluation criteria emerge.

Within this strategic paradigm of design exploration, there are obviously many tactical choices (see Fig. 2).

Finally, there is a battery of *implementation-level* tasks and decisions. Some of these involve AI (e.g., exactly how is knowledge to be represented, what control structure is used, what language or program is employed) and some involve microelectronics (e.g., exactly how will the designs be fabricated).

The "opportunity" mentioned in the title to this section derives from the colocation and interest in collaboration of the creator of the Eurisko program (Lenat), the fabricator of the first few high-rise chips (Gibbons), and someone familiar with both fields (Sutherland). Thus, the implementation-level problems have precise answers The representation and control are taken from Eurisko (frames and agendae), and the fabrication (including design of masks) is performed by Gibbons and his staff. By using the already extant Eurisko program, we were able to concentrate on knowledge rather than on programming and quickly obtain some results. We believed it would be adequate for our task, as it had already discovered concepts and design strategies in other domains.

This article is a report on six months of part-time exploration of this opportunity by the three authors. During this time, we have tried several approaches to questions 1-4, in Figure 2. The rest of this article documents our efforts to date and what we have learned as a result. Just trying to make some of the choices implied above has been most

1 What level of representation is appropriate for primitive microcircuit elements?

2 What kinds of combining operations are needed given the primitive element representation chosen?

3 How are new combinations of elements constructed (e.g., randomly or with some kind of common-sense knowledge about likely usefulness)?

4. What are good criteria for evaluating a new complex structure to decide if this structure should be recorded as useful or thrown out as not good for anything?

Figure 2 Tactical choices to be made in automatic exploration for microcircuit structures.

illuminating. In our discussions and early program runs, we have invented important new structures and design heuristics and improved our understanding of the space of choices.

Experiment 1: Eurisko Applied to Random Generation at the Carrier Level

Our initial idea was to include knowledge about properties of semiconductor regions such as doping, diffusion and drift, recombination, etc. Rules for synthesizing new devices would place different kinds of regions in contact and then interactions would be deduced. We had only some very general evaluation criteria in mind at this point. Gain, non-linearity frequency dependent behavior, etc., seemed like interesting properties that would make a new device promising even before any specific application for it were known.

This first "carrier level" of representation was largely explored by hand. We tried to sort out the primitive elements, rules, and evaluation methods that, as a start, could generate and notice well known devices such as the Junction Diode, Bipolar Transistor, and MOSFET. Since we tailored our primitive components and operators on this basis, indeed, those well known devices were short expressions easily found once we began running the Eurisko program.

The representation we initially used at this carrier level quickly evolved as a result of Eurisko's noticing useful improvements. That is, the Eurisko program monitored how well the representation matched the processing that was going on, and made suggestions, from time to time, of ways in which that representation might be improved. Let us consider an example of this. In the original representation we provided to Eurisko, each individual device was a unit (frame, Being, etc.) one of whose slots was called "`Terminals`." This slot was always so big (i.e., had, empirically, so many entries) that eventually Eurisko finally decided to split it into pieces by defining two useful specializations of this slot—"`MustBeInputTerminals`" and "`XorInputTerminals`" (a list of sets of terminals, such that for each set, one and only one element must be an input terminal).

Below are four concepts as they appeared in Eurisko. The first represents the set of all physical devices; the second is the archetype for an individual device; the third represents a particular individual device; the fourth represents a heuristic rule which takes thermal motion of carriers into account.

```
Name:                SetOfAllDevices
Generalizations:     (SetOfAllPhysicalDevicePhysicsObjects
                     SetOfAllPhysicalObjects
                     SetOfAllDevicePhysicsObjects
                     SetOfAllDevicePhysicsConcepts
                     SetOfAllComplexStructures
                     SetOfAllComplexStructuresBuiltOutOfComplexStructures
                     Anything)
IsA:                 (AbstractDevicePhysicsObject
                     DevicePhysicsConcept
                     AbstractObject
                     SetOfUnits
                     Set
                     Anything)
InitialWorth:        500
Worth:               800
DomainOf:            (Abut DAbut CopyDevice ApplyEField ApplyCEField)
RangeOf:             (Abut DAbut CopyDevice ApplyEField ApplyCEField)
FocusTask:           FocusOnDevices
TypicalExample:      TypicalDevice
Examples:            (TypicalDevice SimpleNRegionDevice SimplePRegionDevice)
MyCreator:           Lenat
MyTimeOfCreation:    "19-July-81 13:37:18"
MyModeOfCreation:    (Copy&Edit SetOfAllShips)
```

Name:	TypicalDevice
IsA:	(SetOfAllPhysicalDevicePhysicsObjects SetOfAllPhysicalObjects SetOfAllDevicePhysicsObjects SetOfAllDevicePhysicsConcepts SetOfAllComplexStructures SetOfAllComplexStructuresBuiltOutOfComplexStructures Anything)
InitialWorth:	500
Worth:	500
PartOf:	Device
Parts:	(SolidStateMaterials EFields Devices)
SimulationHeuristics:	(H65 H66)
FocusTask:	FocusOnTypicalDevice
MyTypicalExampleOf:	SetOfAllDevices
MyCreator:	Lenat
MyTimeOfCreation:	"19-July-81 13:40:55"
MyModeOfCreation:	(Eurisko suggested Copy&Edit TypicalShip)

Name:	Device-817
IsA:	(SetOfAllPhysicalDevicePhysicsObjects SetOfAllPhysicalObjects SetOfAllDevicePhysicsObjects SetOfAllDevicePhysicsConcepts SetOfAllComplexStructures SetOfAllComplexStructuresBuiltOutOfComplexStructures Anything)
InitialWorth:	500
Worth:	600
PartOf:	Device-809
Parts:	((SolidStateMaterials: NRegion-1953 PRegion-75 NRegion-1954) (EFields: EField-930 OrthogonalEField-18) (Devices: no subdevices))
SimulationHeuristics:	(H65 H66)
Terminals:	(NRegion-1953 NRegion-1954)
InputTerminals:	NIL
XorInputTerminals:	((NRegion-1953 NRegion-1954))
FocusTask:	FocusOnDevice-817
MyCreator:	(Task-82 "Find examples of SetOfAllDevices")
MyTimeOfCreation:	"25-July-81 16:02:29"
MyModeOfCreation:	(ApplyEField (ApplyOrthogonalEField (Abut NRegion PRegion NRegion)))

Name:	H65
IsA:	(SimulationHeuristic Heuristic MultiValuedOp SideEffectsOp AbstractOp Op Anything)
UsedInSimulating:	(TypicalDevice)
English:	(If you are simulating a physical device, Then it's important to simulate the thermal meanderings of carriers in the solid state materials in the device)
Abbrev:	(If a device has solid state materials, Then simulate thermal motion of carriers)
IfCurrentTaskIsToWorkOnA:	PhysicalDevice
IfCurrentTaskIsToPerformA:	Simulation

```
IfSimulating:                PhysicalDevice
IfPotentiallyRelevant:       ( λ (dev)
                             (Setq SpaceToUse
                             (TheSubsetOf (Parts dev)
                             (WhichAre 'SolidStateMaterials))))
IfTrulyRelevant:             (λ () (MoreThan1KindOfElement SpaceToUse))
ThenPrintToUser:             Simulated the thermal motion of the carriers in d)
ThenCompute:                 <lisp code that finds the type of carrier for each
                             region, computes the penetration depths into all
                             neighboring regions >
ThenAnalyze:                 <lisp code that analyzes what occurs at each boundary>
ThenFillInEntries:           <lisp code that adds values to dev's Behavior slot>
Arity:                       1
Domain:                      Task
Range:                       (Entries for (Behavior dev))
InitialWorth:                700
Worth:                       750
Generalizations:             (TypicalOp TypicalHeuristic H60)
MyCreator:                   Lenat
MyTimeOfCreation:            "19-July-81 15:11:03"
MyModeOfCreation:            (Copy&Edit H60)
MyLastRunOn:                 SimplePRegionDevice
MyThenComputeRecord:         (12 successes, averaging 72 seconds each)
MyThenComputeFailedRecord:   (1 failure, averaging 4 seconds each)
MyThenPrintToUserRecord:     (12 successes, averaging 9 seconds each)
MyOverallRecord:             (12 successes, averaging 89 seconds each)
MyOverallFailedRecord:       (1 failure, averaging 16 seconds each)
  <a dozen other such record-keeping slots>
```

Although this article is not focusing on representation of knowledge, let us briefly illustrate how new domain-specific kinds of slots are generated by Eurisko We have already seen the usefulness of doing this kind of activity when `Terminals` was specialized to form two new slots, `InputTerminals` and `XorInputTerminals`

In H65, the `IfPotentiallyRelevant` slot used to contain an extra condition not shown above—a predicate testing whether or not the current task (the one chosen from the agenda) dealt with simulating a physical device. So many heuristics had `IfPotentiallyRelevant` slots whose values were "test whether or not the current task deals with simulating X" that Eurisko decided to make that a new slot called `IfSimulating`. So H65 now only has to have an `IfSimulating` slot with the value `PhysicalDevice`. The `IfPotentiallyRelevant` slot could now be shortened (and in many cases completely eliminated). Whenever it was needed (i.e., some rule interpreter asked for a heuristic's `IfPotentiallyRelevant`) an extra test would be synthesized automatically from that heuristic's `IfSimulating` slot.

A second case of forming a new slot happened later when Eurisko noticed that—in several domains, not just VLSI—many of the `If-` slots had constructions of the form `(Setq SpaceToUse X)`. It defined a new kind of slot, `IfSearchSpaceCanBeComputed`, that is simply filled with `X`, and whose side effect is to bind the variable `SpaceToUse` to `X`.

A third example of this process was when Eurisko noticed that many of these slots had values of the form `(Subset (s d) (WhichAre 'y))` The new slot in this case, called `IfSubspaceCanBeComputed`, is simply filled with the list `(s y)`, in this case the list `(Parts SolidStateMaterials)` By this compact entry, the new H65 communicates that it wants to assure that some parts of the device are solid state materials, and assuming this to be the case, binds the variable `SpaceToUse` to the set of parts which are solid state materials.

Note how, as this process of defining new slots goes on, the heuristic gets a few extra slots, but the length of the entries goes down dramatically

This focus on adding new kinds of slots is not a digression, but rather the main new source of power that Eurisko uses AM worked because its representation—LISP predicates—was very natural for the concepts they represented -- characteristic functions for mathematics concepts. This was really a lucky accident due to John McCarthy *designing* LISP to be a natural language for mathematics. Random mutations and compositions of the LISP code often resulted in code that was the characteristic function of an interesting, useful mathematical concept. But LISP is not a natural language for representing heuristics. Encoded as large lumps of LISP code, almost any small change or combination is bound to be disastrous. Above we saw how Eurisko has evolved a natural, well-matched language for stating heuristics compactly—a language that facilitates their discovery and combination, something Conway and Stefik would call a *synthesis language.* Their article in this issue of *The AI Magazine* expands upon that idea

Much of the knowledge in H65 is embedded deeply within

the Then- slots. This is somewhat unfortunate from the point of view of modifying H65 to get new heuristics. H65 computes penetration depths, analyzes each interregion boundary, and decides where annihilation will occur, depletion layers form, etc All this is packed into two slots—ThenCompute and ThenAnalyze. One very general result from Eurisko's work in other domains was that it is more important to finely categorize and partition the If- parts of a heuristic rule than the Then- parts. This appears to hold for our three-dimensional microelectronic circuit design task as well, as there was little urging to split the Then- slots in any way. This seems to be related to the fact that most of the new heuristics synthesized, both in this domain and in others Eurisko has worked on, have had modifications to If- parts—very few successful new heuristics have had modifications to Then- parts. It remains to be seen whether this is a phenomenon to be studied or a defect to be overcome.

The main observations to make from the four units are:

1. Concepts are represented as lists of attributes (slots) and associated values. This structuring allows rules to be very specific yet still remain brief.
2. Some slots are prefaced My- to indicate that they refer to the unit as a data structure. This implements the distinction between object- and meta-level knowledge.
3. Heuristics are represented essentially the same way as all the other knowledge. This enables heuristics to apply to each other as well as to VLSI concepts.
4. The conditions and actions of a heuristic are spread out over many slots. This enables new heuristics to be created as small variants of known ones.
5. A great amount of bookkeeping and recordkeeping is done. This enables later attempts at induction about the knowledge and its use.
6. Separate units are maintained for TheSetOfAllX's, TypicalX, and each X This forces the builders and users of the system to avoid ambiguity.

The basic control structure is that of best-first search. An agenda of tasks is maintained, with symbolic reasons supporting the plausibility of each task. One task looked something like that below:

At each moment, Eurisko is working on the task with the highest priority, which in turn is a number derived from the reasons supporting the task. To work on a task, Eurisko scans through its collection of heuristic rules, finds those which are relevant, and executes (obeys) them. During the execution of a heuristic, three types of actions can occur:

1. new tasks can be proposed and added to the agenda,
2. new concepts can be defined, and
3. new values can be found and added to some slot of some unit.

Although all three actions might occur many times for each task, for the task below (Task-2610) we would expect that sometime while working on Task-2610, during the execution of some heuristic H that was relevant to it, one of H's actions would be to fill in some values for the I/OBehavior slot of the unit called Device-817.

Some slots are filled in when the unit is first created (e.g., Name, IsA, InitialWorth, Parts, MyCreator, MyTimeOfCreation, etc.), some are filled in gradually and continuously as part of recordkeeping (e g., MyOverallRecord), and some are filled in only during the execution of tasks that specifically call for finding those entries (e g., Examples, Specializations).

Experiment 2: Eurisko Applied to Systematic Generation at the Tile Level

To extend Eurisko to discover physically novel devices (e.g., ones with nonlinear gain due to striped doping patterns), we would have to program the various equations involved—equations which are much more complex in form and usage than the trivial ones Eurisko employed, such as the one to compute the thickness of a depletion region. Additionally, we would have to extract and include into the knowledge base many heuristics for when and how to use the equations, what terms to ignore under what circumstances, etc. Doing this was beyond the scope of an initial exploration. Rather, we moved on to a higher level of functional abstraction

We briefly considered a very abstract topological level of representation and rejected it as admitting too many interesting designs that could never be realized geometrically.

```
Name:          Task-2610
IsA:           (SimulationTask DevicePhysicsTask TaskToFind Task Anything)
RunAs:         ((Run 24, Task 180, Task 195)(Run 26, Task 14))
English:       "Find the I/O behavior of Device-817"
UnitToWorkOn:  Device-817
SlotToWorkOn:  I/OBehavior
Priority:      743
Reasons:       ((Device-817 is taking up a lot of room so let's see if it's a loser)
               (Device-817 was recently created and we should gather data on it)
               (The I/OBehavior of Device-809 wanted this to be done))
OnAgenda:      (DevicePhysics)
MyIsA:         (EuriskoUnit)
My...etc
```

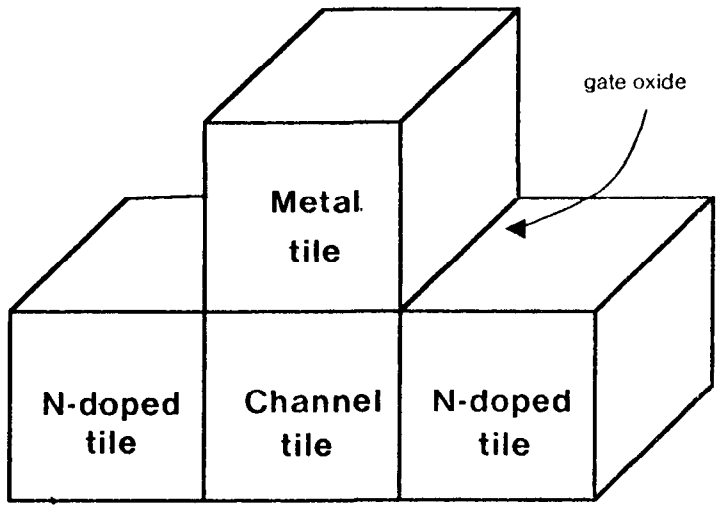

Figure 3 A simple MOS gate If the Metal tile is high, the two *n*-tiles are connected together electrically The "Metal" tile can be any conductor Thus, this gate might be drawn as a standard "red over green" transistor A similar gate exists with the metal tile below, rather than above, the *n-c-n* tiles Two other primitive gates exist, using *p-c-p* rather than *n-c-n* For them, conduction occurs if and only if the metal tile is low

We then settled upon an intermediate level we call the *tile model.* A device is composed of a set of regions. The five types considered were *n*-doped, *p*-doped, channel, insulator, and metal. Each region is conceived as a tile and every tile in the device has the same size and shape For now, visualize a tile as being cubical. These tiles are then packed into three-dimensional arrays and each such array corresponds to a three-dimensional microcircuit

The *n*- and *p*-tiles refer to negatively- and positively-doped substrate material. Channel tiles represent undoped substrate (i.e., more or less intrinsic channel material) which are by default presumed to be coated with gate oxide on both their top and bottom surfaces. Metal tiles represent not just true metal, but any conductor at all (such as polysilicon) which can serve as a gate

Constraints on the building of structures are few. Layers of semiconductor tiles (*p*, *n*, and *c*) alternate with metal layers (metal and insulator tiles). In slightly more detail, our first guess at the possible tiles in the metal layer were `SolidMetal`, `SolidInsulator`, `TopHalfMetal`, and `BottomHalfMetal`. The half and half tiles were used for providing electrical contacts across a substrate tile without shorting to it—that is, each ones serves as a miniature version of a wire. Other types of tiles were experimented with, such as `CenterMetal` and `CenterInsulator`, `MetalThread` (i e., a vertical core of metal surrounded by insulator on all sides), and even diagonal metal. Finally, we decided to conceptualize each metal layer as if it were two separate, independent layers one on top of the other. The only possible tiles in each layer are `Metal` and `Insulator`. To get the effect of `TopMetal`, one places a metal tile over an insulator; to get a `SolidMetal`, one places a metal tile over a metal tile; etc

Since each channel tile is coated on both its top and bottom surfaces with a thin layer of gate oxide, an MOS gate is formed if a metal tile is placed above or below the channel tile, and two *n*- (or two *p*-) tiles flank the channel tile (see Fig. 3).

For electrical behavior, we used a switch level model which connects source and drain tiles when the appropriate gate control signal is present. In terms of tiles, the details are:

> IF you see *n-c-n*, with *m* above (or below) *c*,
> and *m* is "high,"
> THEN the two *n* tiles will be connected together electrically

> IF you see *p-c-p*, with *m* above (or below) *c*,
> and *m* is "low,"
> THEN the two *p* tiles will be connected together electrically

These are somewhat simplified; for example, in the first rule, we should also check that at least one of the *n* tiles is *low.* From the simulator's point of view, this is not necessary, since if both *n* tiles are *high*, that is equivalent to their being connected together electrically (i.e., there is only one idealized voltage level called "high")

Unlike purely topological models, a lattice of tiles cannot be stretched and twisted into unrealistic connectivities. The lattice retains enough geometric reality to permit exploring three-dimensional structures and expect them to be realizable and fabricatable in a straightforward manner. Yet the tile model is simple—it avoids most of the details that bog

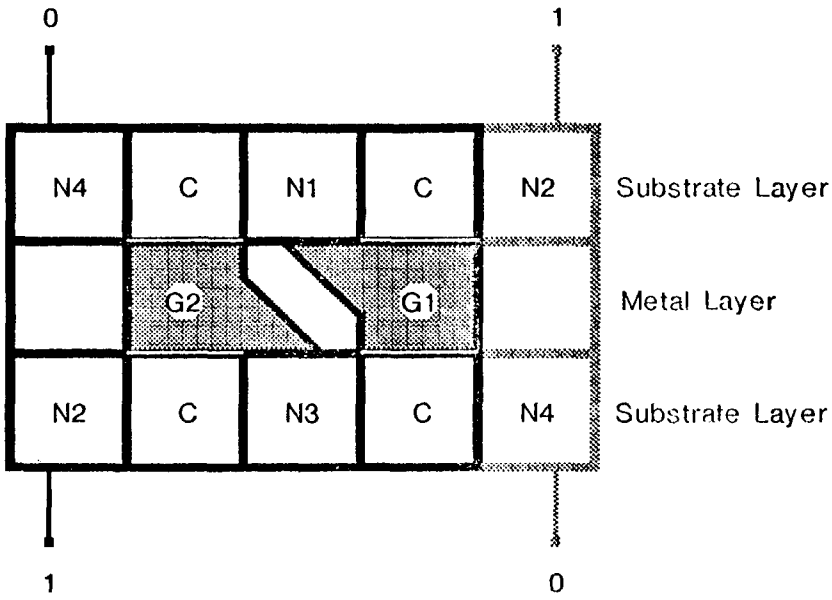

Figure 4 A side view of a device designed by Eurisko that it claimed was a flip-flop The rightmost column was not provided by Eurisko explicitly, but rather was assumed to exist due to a programming bug in array bounds checking, a bug introduced by Eurisko "improving" itself

down any fully authentic geometric model The tile model focuses on the neighbors of a region rather than the details of size, shape, orientation, how to fabricate it, etc.

From another viewpoint, by changing from the carrier model to the tile model, we shifted the domain of exploration from an analysis of potentially interesting semiconductor configurations (interesting based on electrical characteristics such as nonlinearity) to an exploration of the combinatorial possibilities inherent in various arrangements of tiles in a lattice (such as recognizable functionality).

Eurisko was able to handle this shift surprisingly easily. Within a week it was generating and examining arrays of tiles. It soon became clear that we were very poor at visualizing the various devices Eurisko came up with. To aid us, we bought a collection of 1″ square, .25″ thick ceramic shower tiles of various colors Employing them, we realized why Eurisko was (wrongly) claiming the structure in Figure 4 acted like a flip-flop

Due to a programming bug (introduced by Eurisko, incidentally, in its attempts to modify its own code), Eurisko was not always checking its array bounds properly. It thought that the right neighbor of the rightmost column of tiles (in Fig. 4) was the leftmost column of tiles, and not only that but with up-down inverted. This structure is constructible in three-space, namely as the surface we call a Mobius strip. If one builds the device shown in Figure 4, holds it by the ends, gives it a half twist, and fuses the two ends together, the behavior of the device *is* that of a conventional flip-flop, as Eurisko claimed (see Fig. 5). Although it could be built, and although it does use significantly fewer regions than a standard memory cell, given present fabrication techniques it is not a cost-effective design for large-scale production.

We have already covered the answer to decision 1 from Figure 2, that is, the level of representation chosen. As for 2, the combining operations are quite simple in the tile model--one simply stacks up tiles into three-dimensional arrays. Combination translates to adjacency—two devices are combined by pushing them next to each other. The combined device is more than the sum of its parts in three cases:

1 Two metal tiles—one from each subdevice—happen to wind up touching. In this case, a new electrical connection has been made.

2 A metal tile from one subdevice happens to wind up directly above or below a (oxide-coated) channel tile from the other subdevice and there are some doped tiles adjacent to the channel tile.

3 Two doped tiles-- one from each subdevice—happen to wind up touching If they are both of the same type, a new (low-grade) electrical connection is made; if they are of different types a junction diode is formed.

The answer to 3, from Figure 2, is that the combination process was random—often one of the subdevices is a simple single tile.

The answer to 4 is more involved. Question 4 asked how a synthesized device was evaluated. The steps involved here are as follows:

> First, a pass is made through the (unit representing and describing the) device to find all regions that are electrically connected to each other permanently. This can

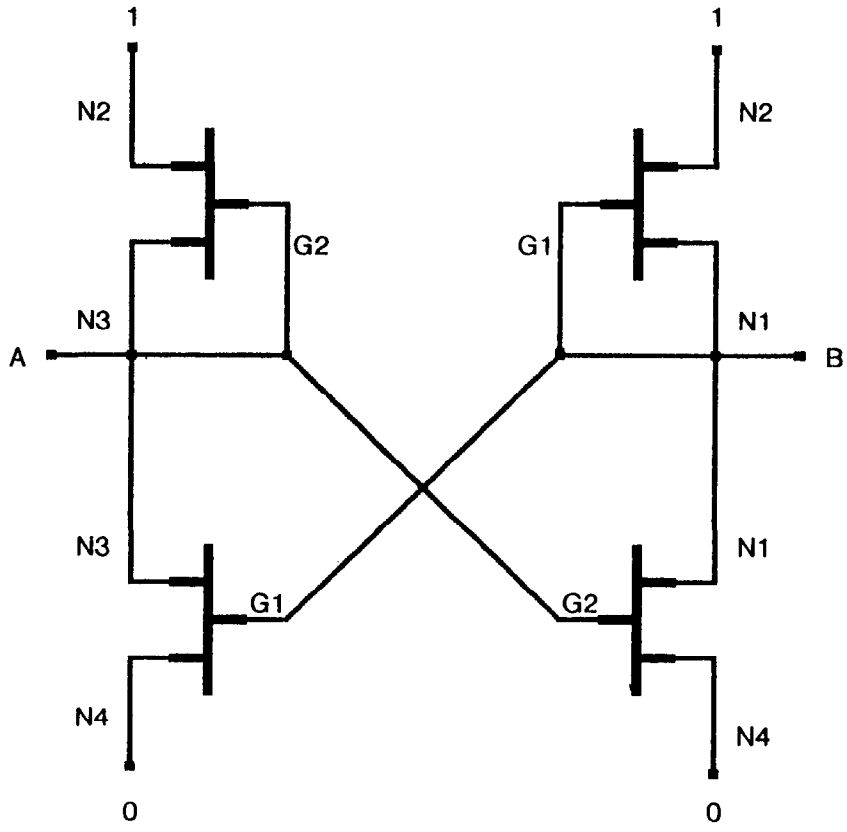

Figure 5 The circuit Eurisko thought the device in Figure 4 was equivalent to

occur within a substrate layer (whenever two like-doped tiles are adjacent) or within a metal layer (whenever two metal tiles are adjacent) or between two layers (whenever a metal tile is above or below a doped tile). By finding the transitive closure of such pairwise connectivities, the entire device is partitioned into electrically connected equivalence classes

Second, a pass is made to find all the possible gates (MOS transistors) in the device. This occurs wherever a metal tile is above or below a channel tile and (at least) a pair of like-doped tiles are adjacent to the channel tile. A rule checks to see if the regions (equivalence classes) which would be connected by this gate (were it "on") are the same. If so, this gate will always be a NO-OP and, hence, can be ignored.

Once the gates are known, the device can be partitioned by the equivalence relation "Might possibly be connected to, by gates" Ideally, one input terminal will exist for each such region If more than one exists, a short might develop, so special care must be taken in those cases. Additional constraints are brought to bear. Finally, a set of all legal divisions of terminals into inputs and outputs is computed.

For a given set of input terminals, all logical inputs are computed and simulated through the circuit. Situations involving "state" require more than one call on the simulator. At this point, (some of) the I/O behavior of the device is known.

The input/output behavior is then "parsed" into larger functional units already known by the system In the case of a device created from subdevices, this behavior will usually refer to at least some of those subdevices. The basic elements initially supplied were logical operators (such as AND), flip-flops, stack cells, light controllers, 7-segment decoders, and a large set of mathematical operations (such as factoring, squaring, unioning, etc.) that were available essentially "for free" as one of Eurisko's earlier domains was elementary mathematics. There was also a remote possibility that the program would stumble onto a device whose behavior could most easily be explained in terms of some Traveller fleet battle operation, or some biological concept, etc., but this never occurred.

Once the description of the device's behavior is at as high a level as possible, it is evaluated by a set of heuristics. These check for such events as the following:

1. computing the same function as X, but in less time or space or power;
2. computing the same functions as X and Y, but in much less than the combined space;
3. symmetry; etc

Thousands of hours of runs with this version of the program (over the course of about one month—yes, we were using multiple Dolphins) convinced us that the "hit rate" for good devices was below one in a billion and gave us a healthy

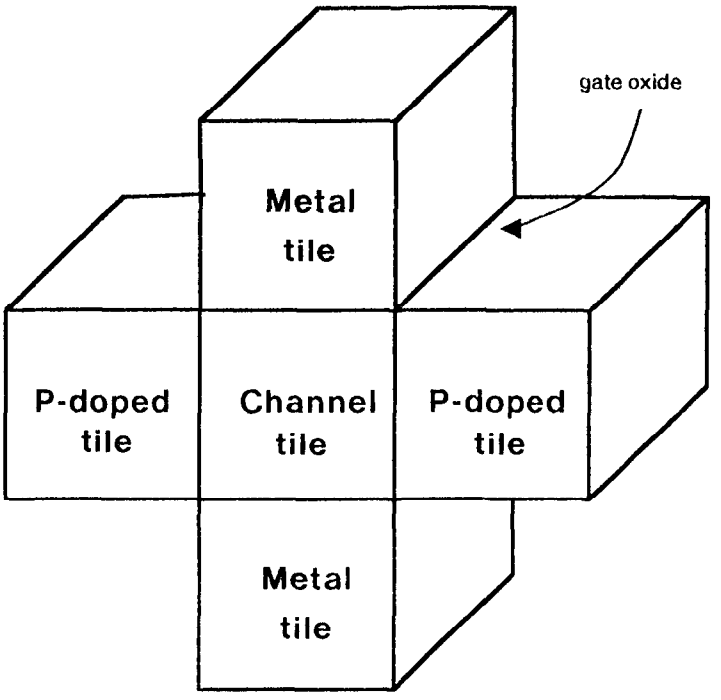

Figure 6 Side view of a gate augmented with a new metal tile to make it more symmetric

respect for the size of the search space for even such small devices as $3 \times 3 \times 3$ ones.

Experiment 3: Eurisko Applied to Best-first Generation at the Tile Level

Blindly searching for interesting microcircuit structures is combinatorially too explosive to be profitable for even very small devices. The solution was to remain true to our paradigm of rule-guided heuristic search—that is, find (either manually or by having Eurisko discover them) some heuristics which could guide the program toward plausible new devices to consider. The legal move space was too large to have merely *implausible pruning* heuristics—most of the generation would have to be constrained by *plausible generation* heuristics. For this experiment, Eurisko remained at the tile level, as described in the last section.

As we are exploring completely new territory, it is "fair" to provide as much help as possible to the program Its final evaluation will be in terms of genuine new discoveries it motivated or made itself. With that in mind, we allowed Eurisko to use all the very general, domain-independent heuristics that it had accumulated from other domains. These included some strategies such as noticing trends and tendencies, augmenting structures to make them more symmetric, examining extreme cases, etc.

Previously, in the unguided search experiment, a new small ($3\times3\times3$) device was synthesized every .9 second. Now, with a hundred heuristics guiding the generation process, it took about 30 seconds to produce each device design. These times are for Xerox 1100's (Dolphins) which currently run a version of Eurisko at approximately 1/4 the speed of Eurisko on a DEC 2060.

Despite the slowdown of 1.5 orders of magnitude, the frequency of valuable new devices rose from one in 10,000 to one in 10. In fact, six of the first twelve devices turned out to be exceptionally valuable. A symmetrizing heuristic was responsible for them. Let us see how they arose.

In the very first case, the heuristic took a highly valued known device—a gate—and tried to make it more symmetric. If you look at the standard gate (Fig. 3), you can see the same obvious addition to make it more symmetric, namely, add one metal tile below the channel tile (see Fig. 6).

This symmetrized structure is quite important. It is an efficient way to compute OR, as the two doped regions will be at the same voltage level if either metal is high. (Recall that all Channel tiles are coated with gate oxide both above and below.)

One of the next few devices to emerge was the same thing as the one depicted in Figure 6, but with *p*-doped semiconductor regions instead of *n*-doped. This is a compact way to compute NAND, as the doped regions are at the same level *unless* both metals are high.

The very next device produced by the symmetry heuristic was the one presented in Figure 7 It is the other, slightly messier, way to symmetrize the gate.

This device is also quite important. It has a single piece of metal controlling two "poles." Many circuits, for example, inverters, employ two gates whose control signals

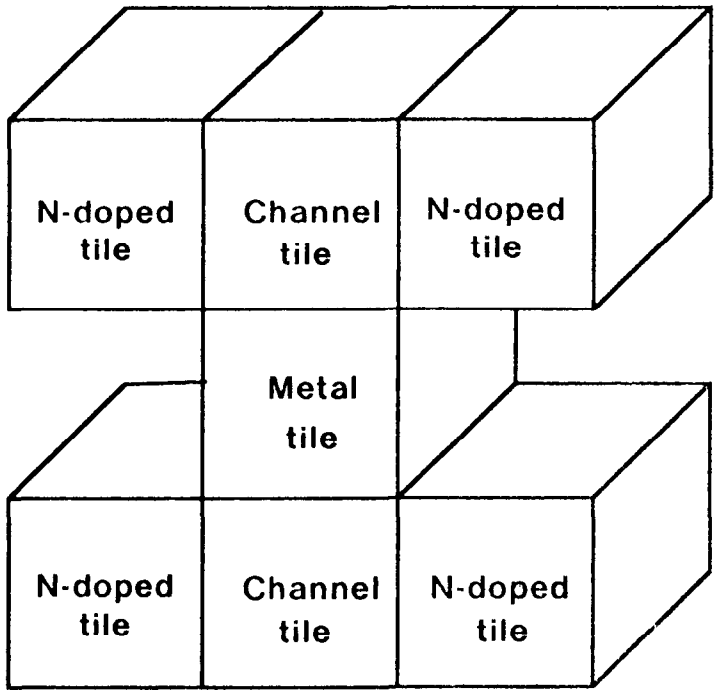

Figure 7 The second way Eurisko symmetrized a gate (side view)

are tied together. In three-dimensional microcircuit design, both of these gates can be realized by the single device above, thereby saving an extra metal tile (for the second gate) not to mention several metal tiles which would have functioned like a wire connecting the two metal gate tiles.

The next device the heuristic produced was a slightly less-preferred symmetrizing, less highly rated because it used p as well as n tiles (see Fig. 8).

This device turned out to have a very interesting behavior. When the metal is high, the two lower, n-doped regions are connected electrically; when the metal is low, the two upper, p-doped regions are connected This device did not surprise Gibbons at all, as he had independently come up with it earlier. It formed the building block for the first high-rise chip ever produced, his one-gate-wide CMOS inverter (Gibbons and Lee, 1980). When the input signal A is high, the lower (n-doped) regions are connected, so the rightmost n-doped region is 0 (low). But the rightmost n- and p-doped tiles are both joined by a metal tile that is also taken to be the output, so in this case the output is low. Similarly, if the input A is low, a channel forms across the top and the output is high

The next device produced by the symmetry heuristic was similar in materials to the one above, but it was a horizontal arrangement of them. Figure 9 provides a view of the device.

When the metal is high, the two n-doped tiles are connected; when it is low, the two p-doped tiles are connected. Note how this exploits the intrinsic nature of the central channel tile capable of supporting a current of electrons or of holes.

The next few symmetrizings were uninteresting The twelfth one took the design from Figure 9 and added a gate underneath it, thus making it more symmetric. This new device, which we call the JMOS cross, is the building block of our current designs, a new design technology we call XMOS (pronounced "cross-moss"). As shown in Figure 10, it can be used to compute both NAND and OR simultaneously and it tesselates three space (it packs side to side and also on top of each other), so that in the long run we get these functions at a cost of just one metal tile, one channel tile, one n-doped tile, and one p-doped tile. It was extremely unintuitive that this could possibly be done at all before we saw this design By not fixing two of the inputs to be 1, as we do in Figure 10, more complex conditional expressions can be computed by these devices.

Conclusions

One important choice for the VLSI design task is the level of abstraction employed. The charge model needs a lot of mathematical back-up to deal with the electrical properties of the component interactions. This model operates relatively closely to natural phenomena with little abstraction The tile model, in contrast, retains enough geometrical detail to keep us honest with respect to fabrication constraints along with enough electrical detail to determine functional utility It ignores enough detail that thousands of carrier-model-level devices map into the same tile-model-level device. But even this is not sufficient in and of itself to allow random or systematic search to be fruitful A few heuristics had to be added to guide the search for plausible devices. Of these, a symmetrizing heuristic had

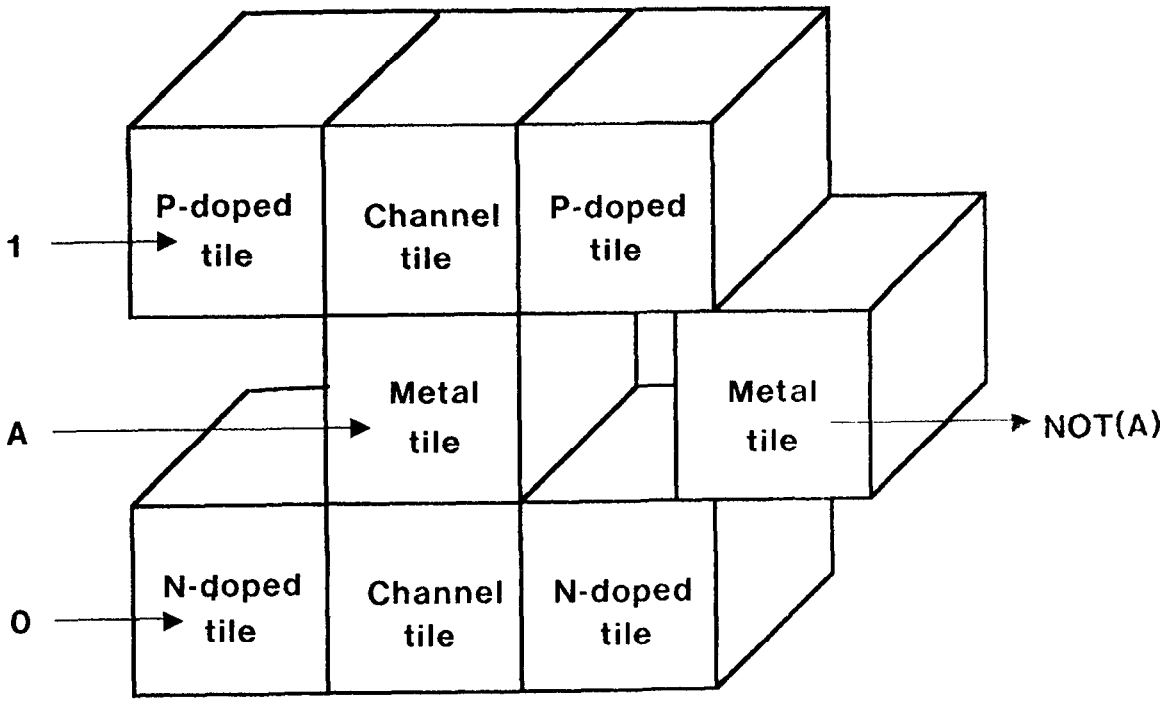

Figure 8 The third way Eurisko symmetrized a gate (side view) The electrical connections shown turn it into a one-gate wide inverter The two metal tiles are not touching each other

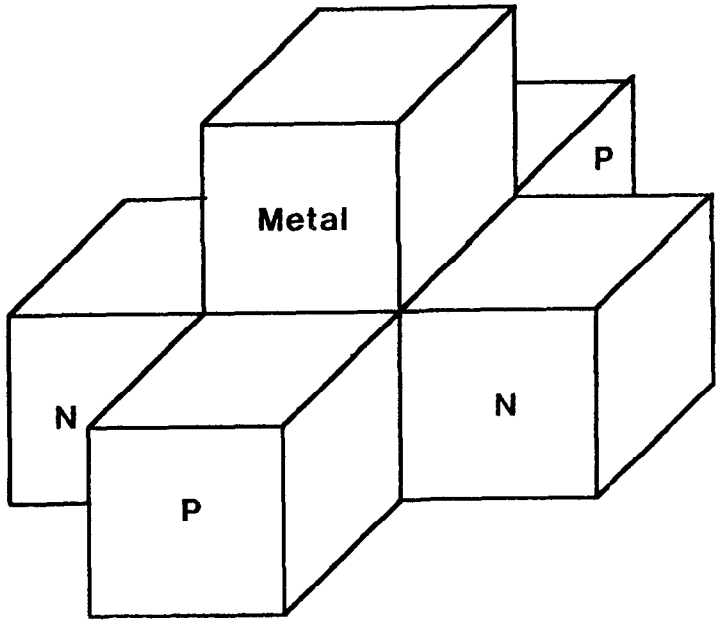

Figure 9 The fourth symmetrizing (side view) The metal tile is laid on top of, and obscures, the Channel tile Either the two *n*-tiles or the two *p*-tiles will be electrically connected, depending on whether the metal tile is (respectively) high or low

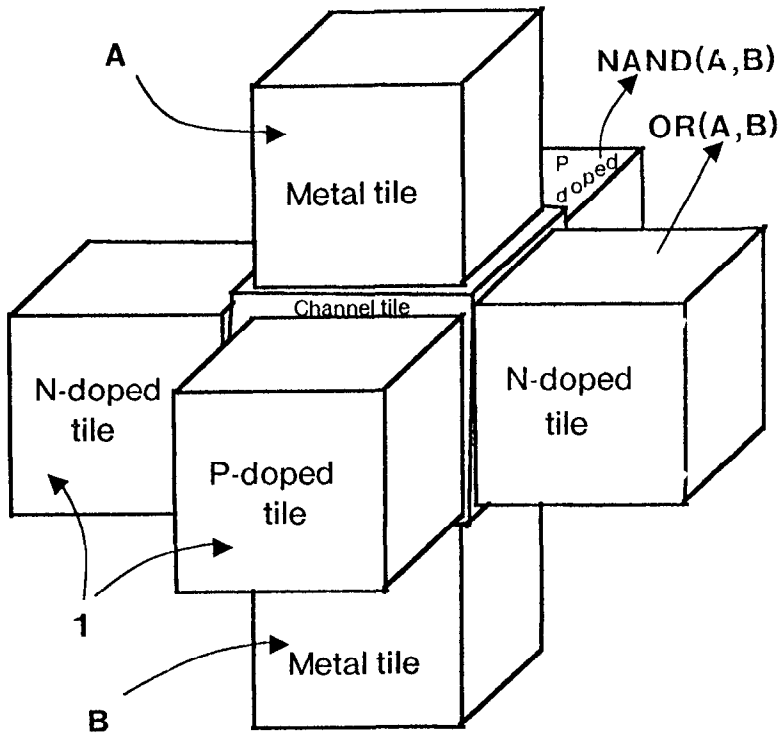

Figure 10 A fifth symmetrizing, related to Figure 9's but with an extra piece of metal added In the center of the device is a Channel tile (almost completely obscured in the figure) The Channel tile lies in the same plane as the four doped tiles

great success—half of its first dozen suggestions turned out to be valuable new devices. In a next try, we will probably try to merge the best features of both the charged carrier and the tile models, perhaps in a sequential way (i.e., worry more carefully about electrical details once an interesting lattice arrangement is found)

The operation of the Eurisko program was satisfactory In this new field, we were able to enter the knowledge for the two different models relatively easily. The existing learning heuristics remained applicable. One of the conclusions from earlier research on Eurisko was the importance of generating new, task-specific kinds of slots, as well as new heuristics. This turned out to be true for the VLSI design task as well, and in this article we have illustrated several cases of automatic and semi-automatic defining of new slot types.

Although it was expected, the futility of exhaustive search –even at "the right level"—was strongly reinforced We do not yet have a satisfactory understanding of appropriate construction heuristics which provide a reasonable hit rate on useful structures. Even our initial attempts at such heuristics paid off handsomely, however, so we are encouraged to continue our investigation.

The discipline of thought required of us in trying this computer application was extremely valuable. In such a new unexplored field, the organization of our thoughts into a computer-digestible form led us *by hand* to the discovery of several new devices and device possibilities which we shall not catalog in this article One example of this was the notion of doping a region only half-way down, thereby leading to a new kind of precharging of devices provided they are not retriggered too quickly.

Just the act of representing knowledge in Eurisko occasionally provided us with novel insights. The use of an array-like data structure led to the notion of a circuit as a lattice of regions (of a few types), rather than the conventional decomposition of it into small devices hooked together by wires Some of Eurisko's designs do have several metal regions in a row, acting like a wire, but most of its useful devices have few if any of these chains, and where they do exit they are short and (most of) those metal regions serve a dual purpose such as acting as a gate.

There is no need to restrict ourselves to cubic tiles, or any sort of rectangular prismatic regions, of course. Going to fish-and-geese might be a bit too far, however Currently Eurisko tesselates space with cylinders whose cross-section is hexagonal. One problem with this is the difficulty obtaining bathroom tiles with that shape, so it is hard to visualize the designs Eurisko comes up with This is not a purely whimsical difficulty Seeing a set of designs, one for each plane, spread out on a flat screen, makes it arduous to trace functionality. Some of this problem goes away by having Eurisko describe what is going on at higher and higher levels of functional abstraction. Unfortunately, by the time it can do this for any given design, it is usually ready to move on to the next one What was needed was a more natural way to visualize the 3-D structures.

To provide that power, we have equipped a doubly-wide Dolphin display screen with a stereoscopic viewer so we can literally see the structure Eurisko is considering at each moment. We considered various methods for 3-D viewing including varifocal mirrors, expensive optical image-fusing setups, oscillator-driven polarizing goggles, etc., and finally found we could get by with an inexpensive fusion device—a first-surface mirror held vertically near the bridge of one's nose Six other "flat" windows on the Dolphin screen display further information to the user—the state of Eurisko's agendae, details of the current task and why it was chosen, details of the current concept(s) being worked on, the heuristics being applied to further the current task, etc.

We were also able to assess the expertise needed to do the job. This expertise is quite wide ranging and includes knowledge about geometry, semiconductor electronics, and fabrication processing.

The most important conclusion is that there are indeed many unintuitive, simple, yet powerful device designs lying "near the surface" in the space of three-dimensional microcircuits Heuristics which suggest plausible changes and combinations appear to be necessary and sufficient to economically find such devices Eurisko appears to be a promising vehicle for exploring this space as it can find such heuristics, even though they may be counterintuitive to human beings.

Future Directions

Our efforts to date have reinforced our initial opinions that this is a fruitful area of application of AI. We have barely scratched the surface and considering the small effort expended believe there is much paydirt to be mined. We offer no claims that we have found the right level of representation or abstraction yet. It is clear that much more exploration by many more people will be required. In fact, it is likely that the interplay of viewpoints from different approaches will be most productive. One such effort is described in Stefik and Conway's article elsewhere in this issue.

We certainly hope to see other levels of representation and abstraction explored. Building an expert system with the knowledge and mathematics needed for dealing with semiconductor properties at an electron/hole level is one clear direction for future work That is, extending Experiment 1 could be profitable.

Another promising direction is to incorporate more knowledge about fabrication processes and equipment. As mentioned before, the computerization of fabrication equipment is making fabrication knowledge and parameters more precise. Additionally, computer controlled equipment could directly use processing commands derived from a knowledge-based process design system. Such a related application of process design as opposed to device design will require a solid knowledge base of semiconductor and related material properties. To give one simple example, a device may be no faster, use no less power, etc. than an old design, yet be highly prized because it requires fewer *masks* to produce.

The novel feature of an AI system working in an emerging field alongside people who are just learning their expertise deserves careful scrutiny. Trying to do this in another quite unrelated field could provide valuable insight about heuristics for learning.

We note again that the traditional paradigm for microelectronics is design, fabrication, and test. These three steps are performed serially—one is completed before the next begins. It is now becoming possible to merge these three. Computer systems process the design data, control the fabrication, and run the tests. By coordinating the programs that do these activities, a real and new integration of the microelectronic construction process is possible. Our little exploration has helped to convince us of this potential. We discussed in the article the possibility of fabricating with the power on, testing (each part of) each layer as it is deposited. Low yield regions might cause rapid redesign of the rest of that layer In exceptional cases the entire layer could be evaporated and tried over again. "Backup" would finally have been pushed not merely to the hardware level, but to the level of fabrication of hardware!

Our final remark is a strategic one on the possibility of major industrial or national impact if this AI application can be successfully pursued. As we enter the era of VLSI technology, there are shortages of critical people, an explosion of design complexity, and increasingly aggravating test requirements, to name only a few of the problems hindering the field. The advent of three-dimensional VLSI technology explodes the magnitude of all of those problems. Any industrial firm or nation which could successfully devote a large number of computation cycles on a sustained basis to intelligent exploration of microelectronic design, fabrication, and test possibilities would certainly be ahead. We enjoy the dream.

References

Dankel, D D., II, (1979) Browsing in large data bases *IJCAI 6,* 188-190

Davis, R., & Lenat, D. B. (Eds) (1981) *Knowledge-based systems in artificial intelligence.* New York: McGraw-Hill

Feigenbaum, E. A. (1977) The art of artificial intelligence: I. Themes and case studies in knowledge engineering. *IJCAI 5,* 1014–1029

Gibbons, J., & Lee, K (1980) One-gate-wide CMOS inverter on laser-recrystallized polysilicon *IEEE Electron Device Letters* EDL-1, 6.

Lee, K F., Gibbons, J., Saraswat, K. C , Kamins, T I., Lam, H. W , Tasch, A T , & Holloway, T. C. (Eds) (1978) *AIP Conference Proceedings*

Lenat, D B (1982a) Eurisko: Discovery of heuristics by heuristic search. Working paper, Computer Science Dept., Stanford University.

Lenat, D. B. (1982b) The nature of heuristics. *Artificial Intelligence*

Polya, G. (1945) *How to solve it* Princeton, N.J.: Princeton University Press