# COMBINING EXPERIENTIAL AND THEORETICAL KNOWLEDGE IN THE DOMAIN OF SEMICONDUCTOR MANUFACTURING

A DISSERTATION

SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE

AND THE COMMITTEE ON GRADUATE STUDIES

OF STANFORD UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

by

John Llewelyn Mohammed

August 1994

© Copyright by John L. Mohammed 1994

All Rights Reserved

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Edward A. Feigenbaum (Principal Advisor)

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

James Plummer (Associate Advisor)

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

J. Martin Tenenbaum

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Paul Losleben

Approved for the University Committee on Graduate Studies:

# Abstract

Symbolic approaches to the diagnosis task have proven their utility in a variety of domains, from medicine to electronic and mechanical engineering. The power of systems based on these approaches derives from their ability to represent and reason with the knowledge of experts. In addition to solving their primary diagnostic tasks, these systems have the additional benefits of being able to explain their findings, and of serving as repositories of corporate knowledge.

We represent diagnostic knowledge concerning semiconductor manufacturing processes in two distinct symbolic forms. We capture experiential knowledge in the form of a network of heuristic causal associations. We capture theoretical knowledge concerning how the manufacturing processes work in qualitative, discrete-action models of the manufacturing operations.

Both kinds of knowledge are applicable to the task of diagnosing manufacturing failures. Each kind has strengths and weaknesses. Causal associations leave many contextual assumptions implicit. In particular, they do not indicate how the causal relationships depend on the structure of the manufacturing plan or on the structure of the devices manufactured. This is primarily due to a lack of representational machinery for describing this information. This leads to "brittleness": an inability to adapt to changes in context. This limitation is a severe one for the semiconductor manufacturing domain, which is characterized by frequent changes in the manufacturing plans and the products being manufactured. On the other hand, causal networks impose almost no limitations on what phenomena can be represented.

Model-based techniques avoid brittleness by explicitly representing how causal relationships depend on aspects of context and by appealing to principles applicable in many contexts. However, the range of phenomena that can be represented is limited by the expressiveness and tractability of the representational machinery and the lack of theoretical knowledge concerning some of these phenomena.

We argue that the strengths and weaknesses of these two kinds of knowledge are complementary. We describe a mechanism, which we call Generic Rules, by which we can integrate the two kinds of knowledge in a synergistic fashion. Generic rules retain the expressiveness of causal networks, but employ the model-based knowledge to express contextual dependencies. The causal associations, qualitative discrete models and Generic Rules are demonstrated by application to the Stanford BiCMOS process.

# Acknowledgements

Many people have helped me in the course of this work, either by providing needed resources or technical assistance, by engaging in interesting discussions, by reading and commenting on early drafts of the dissertation, or by providing much needed moral support. All deserve my gratitude. I list here some among them that I would especially like to thank.

For their advice, encouragement and support, I wish to thank the members of my reading committee: Professor Ed Feigenbaum, Professor James Plummer, J. Martin Tenenbaum, and especially Paul Losleben, without whose prodding the work may never have been completed.

Reid G. Simmons, whose Gordius system forms the core of the model-based reasoner, and who collaborated in the development of the early versions of the models and the diagnostic reasoner.

Rick Reis, for his advice and encouragement and the financial support provided through the Fellow/Mentor/Advisor program, and Barbara Hayes-Roth, who on several occasions provided much needed resources.

John Shott, Wes Lukaszek, Peter Griffin and Margaret Prisbe, for unselfishly contributing their expertise in semiconductor manufacturing.

The members and former members of the Manufacturing Science program within the IC Lab and personnel from Enterprise Integration Technologies and Texas Instruments who participated in it, including Jay Glicksman, Jeff Pan, Fah-Chun Cheong, Byron Davies, Robert Hartzell, Ernest Wood, Don Basile, William Wong, Jack Wendstrand and others, for numerous discussions regarding representation of semiconductor processes. Also, the members of the CFI TCAD SPR working group, including Duane Boning and Mike McIlrath

The members and former members of the Heuristic Programming Project within the Knowledge Systems Laboratory, including Peter Karp, Richard Keller, Tom Gruber, Yumi Iwasaki, Pandurang Nayak, Janet Murdock, Bob Engelmore and Richard Fikes, for discussions regarding model-based knowledge representation and reasoning.

The many administrative personnel at the KSL and CIS who kept these organizations running smoothly, and who graced them with a touch of warmth and human kindness while doing it, including Grace Smith, Linda Kovach, Michelle Perrie, Susan Stout, Peche Turner, Carmen Miraflor, Ellie Engelmore and Margaret Timothy.

Colleagues at the former Schlumberger Palo Alto Research center, who contributed to the early phases of the work, including J. Martin Tenenbaum, Richard O. Duda, Harry Barrow and Jeff Pan.

The personal friends and colleagues who helped me to maintain my sanity and keep a sense of perspective, including Ed Pednault and Marla Babcock, Fred and Carol Shapiro, Margaret and Dennis Donohoe, Yvan Leclerc and Susy Davies, Roberto and Alison Desimone, Richard Cohn, John Egar, Lee Iverson and Alison Phinney, and David and Mary Kashtan. Especial thanks go to Yvan Leclerc for making much needed computer resources available for the evaluation phase of the work.

Professor Steven W. Zucker, for inspiring me to pursue graduate work in the first place, and for being a constant source of encouragement.

My mother and father, for giving me, through their love and encouragement, the freedom and confidence to live my own life.

Finally, I would especially like to express my gratitude to my fiancée Nancy J. Lerman, whose love and support have made it possible to complete the work.

This research was supported under the Stanford Semiconductor Manufacturing Program supported by Grant N00014-90-J-4016, DARPA/CSTO.

# Table of Contents

Abstract iv

Acknowledgements vi

List of Tables xi

List of Figures xii

Chapter 1. Introduction 1

1.1. The Task: Diagnosis of Semiconductor Manufacturing Problems .............. 2
1.2. The Symbolic Approach to Diagnosis .............. 3
1.3. Problem Statement .............. 5
1.4. Contributions of the Research .............. 6
1.5. Overview of the Dissertation .............. 7

Chapter 2. Background: Semiconductor Manufacturing 9

Chapter 3. Experiential Knowledge 13

3.1. What is Experiential Knowledge? .............. 13
3.2. Representing Experiential Knowledge .............. 14
3.3. HyperPIES .............. 16
3.4. Strengths and Weaknesses of Experiential Knowledge .............. 21

Chapter 4. Theoretical Knowledge 27

4.1. Model-Based Reasoning .............. 27
4.2. Special Considerations of the Manufacturing Domain .............. 29
4.3. The Modeling Technology .............. 31
4.4. Symbolic Qualitative Modeling of Semiconductor Manufacturing .............. 47
4.5. Summary .............. 72

Chapter 5. Process Representation 75

5.1. Representing the Process for DAS ..... 76
5.2. Representing the Process for Display ..... 81
5.3. The DIS Semiconductor Process Representation Server ..... 85
5.4. Extracting the DAS representation from DIS ..... 88

Chapter 6. Architecture for Integration 108

6.1. Phenomena Scope and Situation Scope ..... 108
6.2. Generic Rules ..... 111
6.3. Impact of Generic Rules ..... 118
6.4. System Architecture ..... 119

Chapter 7. Experimental Results 123

7.1. The Experiments ..... 123
7.2. The Results ..... 133
7.3. Summary of the Results ..... 145

Chapter 8. Related Work 146

8.1. Model-Based Reasoning ..... 146
8.2. Combining Rule-Based and Model-Based Reasoning ..... 149
8.3. Modeling Manufacturing ..... 151
8.4. Semiconductor Process Representation ..... 157
8.5. Model Selection ..... 159

Chapter 9. Conclusions 163

9.1. Future Work ..... 163
9.2. Summary and Contributions ..... 165

References 168

Appendix A: Modeling Objects 177

Appendix B: Modeling Actions 182

Appendix C: Axioms Supporting the Models 214

Appendix D: AESOP Causal Associations 223

Glossary ........ 223
Causal Associations ........ 224

Appendix E: Generic Rules 225

Appendix F: Associations from Generic Rules 237

Legend ........ 237
Category 1 Causal Associations ........ 239
Category 2 Causal Associations ........ 246
Category 3 Causal Associations ........ 247

# List of Tables

7.1. Recovery of AESOP associations by Generic Rules ........................................ 139

# List of Figures

3.1 HyperPIES top level control panel ........ 18
3.2 Traversing Fault Concept Taxonomic Hierarchy ........ 18
3.3 Fault Concept Editor ........ 19
3.4 Causal Link Editor ........ 20
3.5 PIES Causal Analysis Graph ........ 21
3.6 Configuration just before Poly Gate Etch ........ 23
3.7 Configuration after Poly Gate Etch ........ 24
3.8 Formation of MOSFET ........ 24
4.1 Example type definitions ........ 39
4.2 Schema for describing processes (actions) ........ 41
4.3 Sketch of Vertical Cross-Section Through BiCMOS Transistors ........ 49
4.4 Cross-sectional drawing of a Collector-Under-Oxide resistor, and how DAS represents this same structure ........ 51
4.5 The basic operations modeled in DAS ........ 56
4.6 The Basic Etch Model ........ 61
4.7 Abstract model inherited by models in which layers can be changed ........ 62
4.8 Qualitative example of a general Etch-Rate rule ........ 63
4.9 Model of End-Detect-Etch ........ 66
4.10 Model of diffusion across polysilicon material boundary ........ 70
4.11 Definition of Create/Modify-Layers models ........ 71
5.1 Simple example of a mask description. ........ 79
5.2 Instance of an Oxidation operation ........ 81
5.3 Semiconductor Process Representation Server scenario ........ 87
5.4 Partial Step Taxonomy ........ 93
5.5 Initial decomposition of semiconductor process plan ........ 94
5.6 CIM versus TCAD notion of “step” ........ 95
5.7 Breakdown of part of Isolation module ........ 97
5.8 Language-Independent Semiconductor Process Representation Server ........ 103
6.1 Graph of Phenomena Scope vs. Situation Scope ........ 110

6.2 Syntax for Generic Rule ........................................................................ 114
6.3 Interlevel Generic Rule for POLY-GATE-UNDER-ETCH example .............. 116
6.4 Intralevel Generic Rule for POLY-GATE-UNDER-ETCH example ............... 117
6.5 Impact of Generic Rules in Phenomena/Situation scope space ..................... 118
6.6 Architecture for integration using Generic Rules ............................................ 120
8.1 Semiconductor Computer Tools as Information Transformers ...................... 155

# Chapter 1. Introduction

There has been much recent interest and activity in applying the technology of “Expert Systems” to engineering domains, including the domain of Semiconductor Manufacturing [1] . The power of a system based on this technology lies in its use of the experiential knowledge of people who are expert at the task for which the system is being constructed.

Some of the limitations of this technology are now well known. A number of these limitations can be traced to a single phenomenon: that the rules encoding the knowledge often implicitly encode a dependence on the context in effect when the rules were written. The knowledge contained in a system for diagnosing processing problems in the semiconductor fabrication domain, for example, can implicitly depend on the manufacturing plan used in the process being diagnosed and the structure of the devices manufactured (especially those devices, called test-structures, used to obtain the manifestations of the problems). These dependencies render much of the knowledge impotent when the context changes, i.e., if the manufacturing plan changes or if the test-structures change.

Modern manufacturing systems require adaptability to rapidly changing environments. In particular, frequent changes in the products, processes, or equipment in a semiconductor processing facility reduce the value of experiential models used in more stable environments. Since the rules in an experiential knowledge base often do not describe the rationale that justifies their inclusion, knowledge bases constructed of such rules cannot be automatically adapted to changes in the environment that can invalidate many of the rules. Thus, maintenance of experiential models is a difficult and labour-intensive task.

Engineers temper and adapt their experiential knowledge by recourse to theoretical knowledge. For other engineering domains, some of this theoretical knowledge has been successfully captured in the form of declarative causal models [2,

3]. These models allow much of the context to be represented explicitly, thus allowing changes in the context to be incorporated in a natural way.

The work reported on in this dissertation is concerned with the representation of both experiential and theoretical knowledge bearing on the problem of end-to-end diagnosis of semiconductor manufacturing processes. It also addresses the strengths and weaknesses of these two types of knowledge. In particular, this dissertation argues that their strengths and weaknesses are complementary, and presents a methodology for combining them that enables the strengths of each to overcome the weaknesses of the other.

## 1.1. The Task: Diagnosis of Semiconductor Manufacturing Problems

Semiconductor Technology is a broad and complex field encompassing many activities. While the technologies to be discussed in this dissertation are applicable to many of the reasoning tasks connected with these activities, this work focuses on one particular task: end-to-end diagnosis of problems in semiconductor manufacturing processes.

Semiconductor manufacturing processes are the processes by which integrated electronic circuits are manufactured. Integrated circuits are manufactured on the surface of thin “wafers” of silicon or some other semiconducting material.

Semiconductor manufacturing plans typically involve a long sequence of operations. Modern processes may have roughly 300 steps that are carried out on every product wafer. The adjectival phrase “end-to-end” is included in the problem description to emphasize that we will be concerned with diagnosing the entire manufacturing process as a whole. We will not be concerned with diagnosis, monitoring or control of individual processing operations or manufacturing equipment. Our focus will be on disentangling the many varied ways that the processing operations can interact.

New manufacturing processes are constantly in development. Furthermore, semiconductor processes are continually modified to improve yield, to adjust for drift in the characteristics of processing equipment, and/or to improve device characteristics. In this work we assume that the manufacturing process as specified is correct. We are not concerned with debugging new manufacturing plans, but in debugging problems in the execution of “correct” plans. It is a major focus of the work to enable the knowledge we

encode to adapt to changes in the manufacturing plan. Nonetheless, the knowledge relates to attributing failures in the product to failures in processing, not to debugging problems in the design of the processing plan.

The sequence of operations in a semiconductor manufacturing plan does not completely specify the structures that are constructed on the wafer. The set of structures defined on the wafer are determined by an input to the plan: the set of *photolithography masks*. These masks are reticles of glass coated in a pattern with an opaque substance, often a metal. The patterns in the masks determine how different areas of the wafer surface are differentiated from one another to form devices and circuits. The masks generally include patterns for the creation of special structures, called *test structures*. These structures are designed so that their electronic behaviour is determined as much as possible by a narrow set of process variables. They are included on the wafer so that the quality of processing can be characterized, and so that errors in processing can be isolated. Our work assumes that the devices and test structures are properly designed.

Finally, there is one additional caveat not alluded to in the brief description of the problem domain. We are concerned primarily with the behaviour of the manufacturing process, less so with the behaviour of the manufactured product. That is to say, we focus on seeking causes for anomalies in the physical structure of the product in terms of anomalies in processing. This is to be contrasted with identifying causes for anomalies in the behaviour of the product (e.g., the electronic characteristics of the devices, the electronic behaviour of the circuits) in terms of anomalies in the physical structure of the product. The latter problem is addressed more fully in the work of Greg Freeman [4, 5] .

## 1.2. The Symbolic Approach to Diagnosis

The characteristic that differentiates the symbolic approach to diagnosis from other computational methods applicable to the same problem is the reliance on declarative representations of knowledge. That is, knowledge of the problem domain in which diagnosis is to be performed is not embedded in the design of a program, but is explicitly represented in symbolic data-structures that can be communicated between the computer and its users, and manipulated by the program. At some level of description, the symbolic data-structures can be interpreted as statements of knowledge about the domain and/or about how to perform the diagnostic task. There are well - defined rules for how these data-structures can be manipulated, and these

manipulations correspond to various forms of logical or probabilistic inference. Overall, the manipulation of the data structures is called reasoning.

The ability to communicate knowledge is a key feature of symbolic systems. The knowledge that symbolic systems use is obtained from the users of the system through such communication. Also, the system can refer to its knowledge to explain the reasoning behind its findings. The accessibility of the knowledge makes symbolic, knowledge-based systems useful for such functions as training, technology transfer, and maintaining a corporate memory.

Other computational tools useful in diagnosis include statistical data exploration tools, statistical characterization techniques, and quantitative simulators.

Statistical data exploration tools provide several ways of attempting to identify the features of the system being diagnosed that best discriminate between success and failure cases. These tools have no knowledge about the system being diagnosed. The user employs his/her knowledge about the system being diagnosed to decide which of possibly many hundreds of features are worth examining. Usually, only a relatively small number of variables can be examined at a time. Therefore, the efficiency with which these tools can be used for diagnosis is directly proportional to the amount and quality of knowledge that the user can bring to bear.

Statistical characterization techniques employ experiments with a working version of the system to be diagnosed, to capture a succinct representation of the system's input/output relationships. The experiments perturb the inputs in controlled ways and record how these perturbations affect the outputs. The resulting input/output characterizations are statistical models of the system. These techniques treat the system as a "black box," and assume that the relationships between the inputs and outputs of the box are relatively simple. When this assumption is invalid, then the characterizations are only accurate within the small space of variations for which experiments were performed. Also, the number of inputs and outputs that the system is considered to have must be kept relatively small. Otherwise the techniques require too many experiments and the resulting models are too complex. It can be considered that statistical models capture experiential knowledge about the system in a mathematical form.

Quantitative simulators provide accurate predictions of the behaviour the system being simulated over a wider range of conditions, and are based directly on theoretical knowledge about the system. However, this knowledge takes the form of mathematical models that are embedded within the simulator program. Unlike symbolic systems, a quantitative simulator cannot explain the reasons for the results it predicts. The simulator behaves just like the real world system it simulates, and is just as inscrutable. For this reason, it is difficult to use quantitative simulators to compute the "inverse transformation": from effects to causes. One must guess at possible causes, and then test the guesses by forward simulation.

All these technologies have a role to play in the complex task of diagnosing semiconductor manufacturing processes. Although there is a tendency among people to consider them as competitors, they fulfill complementary needs. In this work we focus on the symbolic approach to diagnosis.

## 1.3. Problem Statement

This thesis is about representing knowledge concerning semiconductor manufacturing that can support symbolic end-to-end diagnosis of stable semiconductor processes in a manner that enables automatic adaptation of the knowledge base to new processes and test structures.

Previous work in this domain has relied on experiential knowledge captured in hand-crafted heuristic rules. The main drawback of this approach has been that such knowledge is not amenable to automatic adaptation to new contexts. Thus, the principle concern of this work is to address this drawback: to represent the knowledge in a fashion that is robust in the face of changes to the manufacturing process and/or to the structures used to characterize and test the success of the process.

Our solution to this problem has two principal aspects:

1) Representation of theoretical knowledge concerning semiconductor manufacturing in the form of declarative, causal models. These models must be capable of assimilating a description of the process to be diagnosed and descriptions of the test-structures in the form of the masks used to create them, and creating from these a representation of the causal pathways of the process that can support diagnostic reasoning.
2) Representation of experiential diagnostic knowledge concerning phenomena that lie outside the realm of the theoretical knowledge captured in our models. To

enable adaptation of this knowledge to new processes and test-structures, we develop a new method for the integration of experiential knowledge and theoretical knowledge. This method enables the theoretical knowledge to be used to describe how the inferences sanctioned by the experiential knowledge depend on the manufacturing plan and the test-structures.

Although we are motivated by a particular task, that of diagnosing manufacturing problems, our main focus is the representation of knowledge and the content of that knowledge. We will not be concerned with diagnostic strategies.

## 1.4. Contributions of the Research

This thesis makes the following contributions:

- It demonstrates that by careful selection of the appropriate level of abstraction and granularity, it is possible to model semiconductor manufacturing processes symbolically, thus making it possible to capture the causal and functional dependencies in these processes.
- It does so by exhibiting an ontology for describing the structures created on the wafers during processing that captures their geometry in one dimension and their topology in two dimensions, and by exhibiting temporally discrete, qualitative, declarative models of the 12 major classes of manufacturing operations that affect the geometry and topology of these structures.
- It contributes to the development of a unified Semiconductor Process Representation. In particular, it identifies a key problem to be solved by such a representation: extraction of application-specific representations, and describes an approach to solving this problem.
- It introduces two new concepts for characterizing knowledge representation paradigms: Phenomena Scope and Situation Scope. These concepts facilitate comparison of the strengths and weaknesses of different approaches to representing knowledge. In particular, they help to demonstrate that the key strengths and weaknesses of experiential and theoretical knowledge representations are complementary.
- It introduces a new method, called Generic Rules, for synergistic interaction between experiential and theoretical knowledge representations, in particular between heuristic causal associations and behavioural models, that helps to overcome the limitations of each approach taken separately.
- It describes an implemented system, called Language-Independent Semiconductor Process Representation Server (LISPRS), for extracting a process representation suitable for the theoretical reasoner from the Stanford Specification Editor representation stored in the Distributed Information Server (DIS).
- It describes an implemented system that incorporates models of semiconductor manufacturing and generic rules for causal associations that can diagnose manufacturing failures in a modern, complex BiCMOS process.

- It describes the results of attempting to transfer the knowledge in an experiential knowledge base developed for the Stanford CMOS process into a knowledge base suitable for the Stanford BiCMOS process, by encoding the knowledge in Generic Rules.
- It describes the results of attempting to diagnose several processing failures using the aforementioned models and generic rules.

## 1.5. Overview of the Dissertation

In Chapter 2 the semiconductor manufacturing domain is described in greater detail. Some of the key features of the domain are pointed out that contribute to the domain's complexity and to its suitability for symbolic reasoning.

The subject of Chapters 3 is the representation of experiential knowledge. Since the concept of experiential knowledge is somewhat fuzzy, a working definition for experiential knowledge is attempted. Heuristic rules are described as the means of representing experiential knowledge. The concept of causal association is then introduced, and the HyperPIES system, which serves as the exemplar of heuristic rule-based representation of experiential knowledge for this work, is briefly described. Finally, the strengths and limitations of experiential knowledge are discussed, and illustrated with an example causal association.

Model-based reasoning is introduced as a means of representing theoretical knowledge in Chapter 4. Special considerations of the manufacturing domain as it relates to model-based reasoning are described. The features of the Gordius system, the representational machinery we use to represent theoretical knowledge, are described in some detail. Finally, the manner in which this machinery is used to represent wafer structures and semiconductor manufacturing operations is presented. As this is a very long chapter, a summary of the main conclusions of the chapter is included.

The theoretical knowledge can adapt to new manufacturing plans because the plan is an input to the reasoner. This implies that a semiconductor process representation is needed. The issue of semiconductor process presentation is discussed in Chapter 5. This includes a description of the representation required by the theoretical reasoner, and a discussion of how this representation can be extracted from a unified process representation intended to serve all applications that require information about the process plan.

The instrument of our method for combining experiential knowledge with theoretical knowledge is a representation called the Generic Rule. The Generic Rule is introduced in Chapter 6. The comparative strengths and limitations of experiential and theoretical knowledge are reviewed. This comparison is facilitated by the introduction of the Phenomena Scope and Situation Scope concepts. These concepts are also used to evaluate the impact of encoding knowledge in generic rules. Finally, the architecture of a system that employs generic rules is described.

Two experiments are described in Chapter 7. The first experiment is an attempt to transfer experiential knowledge developed for one process to a knowledge base applicable to another process, using generic rules. The second experiment attempts to use both the transferred experiential knowledge and the theoretical knowledge captured in models to diagnose processing failures. The results of these experiments are presented and analysed.

Related work is described in Chapter 8. A summary of the main points of the thesis and a discussion of possible directions for future work are included in Chapter 9.

Parts of this work have been published elsewhere. In particular, [6, 7] present early versions of the models of manufacturing operations, and describe how they can be used simulate manufacturing processes and to perform diagnoses of manufacturing problems. For this reason, Chapter 4 incorporates these papers by reference. The discussion in Chapter 5 of problem of extracting application-specific process representations from the unified process representation is based on a similar presentation of this problem in [8] . Finally, Chapter 5 avoids a lengthy discussion of the representation of masks by incorporating [9] by reference.

# Chapter 2. Background: Semiconductor Manufacturing

Semiconductor manufacturing refers to the manufacture of integrated electronic circuits. These are complete circuits, containing as many as a million transistors, which are fabricated at the surface of a single, thin wafer of a crystalline form of a semiconducting material such as silicon. Each transistor occupies a microscopically tiny area of the wafer's surface, and all the transistors are manufactured simultaneously. In fact, it is usually the case that tens to hundreds of copies of the circuit are manufactured on a single wafer, which is then diced into die containing a single circuit each. The parallelism does not end there: many of the manufacturing operations are carried out simultaneously on several wafers (as many as hundreds) that form a "lot."

The process by which this technological miracle occurs is a mixture of techniques analogous to printing and cooking. Most of the operations involve chemical reactions in chambers, often at high temperatures, or in baths of liquid reagents. These reactions operate across the entire surface of the wafer, to grow extensions to the crystal, deposit films of other materials on the surface, grow oxides of the materials at the surface, incrementally remove materials by etching, or redistribute impurities within the crystal by diffusion.

One of the ways that impurities are introduced into the wafer is by accelerating ions of the impurity in an electric field so as to implant them into the wafer surface. There are two classes of impurities, called dopants, that are introduced. These are elements from the groups III and V of the periodic table. Though semiconducting materials are poor conductors when they are pure, their conductivity improves when impurities from these two groups are introduced into the crystal lattice. Two different mechanisms for conducting current occur. When group V elements such as arsenic or phosphorous are incorporated into the lattice, they afford the crystal unbound electrons for conduction, and the crystal is said to be *n*-type. When group III elements such as boron are introduced, they afford the crystal a surplus of "holes" (covalent bond sites

lacking an electron) and the material is said to be *p*-type. (The terms derive from the charge of the carrier of current: negative for electrons and positive for holes.) When n- and p-type regions are adjacent in the wafer, diodes and bipolar transistors are formed. The type of a region can be temporarily reversed by an electric field that repels one type of carrier and attracts the other. This is the principle employed in the field-effect transistor (MOSFET).

With few exceptions, all operations act homogeneously upon all parts of the wafer surface. The primary way to create structures at the surface that differ from region to region is to employ a photo-optical process similar to photography, called photolithography. The lateral variations in processing desired must be reduced to a sequence of binary patterns. These patterns are photographically reproduced as alternately opaque and transparent areas on glass plates called *masks*. The wafer is coated with an organic, light-sensitive material called photoresist. Then the pattern on the mask is transferred to the wafer by shining a high intensity, high frequency light through the mask, exposing the photoresist in only in those areas that are transparent on the mask. This either has the effect of hardening the resist in those areas (if the resist is a *negative* resist) or softening it (if the resist is a *positive* resist). The other exceptions are directed-beam processes that eliminate the need for a physical mask by using an electronic representation of it to vary the energy of a beam as it is scanned across the wafer surface.

The exposed photoresist can then be developed. Development selectively removes the resist in areas where it is soft. The remaining resist can then protect the wafer against the impact of some processes, notably etching and implantation of dopants. The photoresist could not survive the high temperatures employed for most other operations, and it contains impurities that would ruin the characteristics of the devices to be created if they were permitted to diffuse into the wafer crystal. Thus, photolithography is often used in connection with etching to reproduce the pattern of the mask in a layer of material on the wafer that can effectively mask the effects of these other operations. Such layers are often completely removed after they have served their purpose, and so are termed “sacrificial.”

It is a fact of life in semiconductor processing that almost every operation has unintended, and often undesirable side-effects. The implantation of dopants can damage the structure of the crystal at the wafer surface. All high-temperature operations,

whatever their intended purpose, cause further diffusion of any dopants already present in the wafer. Growing an oxide can rob the wafer of some dopants, and cause other dopants to “pile up” at the surface. The oxidation process can also change the diffusivity coefficients that determine how much diffusion of dopants occurs. Temperatures required for diffusion and oxidation can exceed the melting point of the metals used for interconnecting wires.

These characteristics make semiconductor manufacturing a very complex process. There are two aspects to this complexity. The first is that each operation is a complex chemical process that must be carefully designed and carried out, and that is difficult to model accurately. The efforts at creating accurate quantitative numerical simulators are aimed squarely at overcoming this complexity. There have also been many applications of expert systems technology to the problems created by this complexity. Systems have been developed for diagnosis, monitoring and control, and design of unit processes and equipment. Some of these are reviewed in [1] .

The second aspect of the complexity of semiconductor manufacturing stems from the fact that they involve a large number of steps. All the structures created on the wafer at any given point in the process are potentially affected by all subsequent steps. The impact of a step on any part of the wafer depends critically on what structures are present on the wafer when the step is executed. Many structures are created on the wafer solely to influence the impact of intermediate steps in the process, and are later removed.

In this work we consider how knowledge-based systems can help in handling this second aspect of the complexity of semiconductor manufacturing. The complex interactions between steps in the manufacturing process are well-known to process engineers. However, for a long manufacturing process it can be difficult for a person to keep track of all the details of the interactions that take place. Further, it can require a great deal of time for even an experienced engineer to aquaint himself or herself with the details of a new process. Recording and managing large amounts of information is a forte of computer systems. Knowledge based systems can serve as a repository of knowledge about these interactions, and they can systematically consider all relevant interactions in the course of diagnosing a manufacturing problem.

In other ways, the semiconductor manufacturing domain is less complex than other domains. Semiconductor manufacturing plans are mostly linear sequences of operations. Unlike many physical systems, feedback and feedforward are not present to any significant degree. Also, apart from rework loops in photolithography modules, (which can be safely ignored for the most part) the plans have none of the recursions and iterations that complicate reasoning about software programs. The parallelism inherent in semiconductor manufacturing is strictly lock-step. There is no need to reason about temporal interactions between concurrent asynchronous processes.

The problem of diagnosing manufacturing problems in this domain is complicated, however, by the fact that the semiconductor industry is characterized by fast-paced development. Advancing the state of the art is the key to remaining competitive in this industry. New devices, circuits and manufacturing processes are continually being designed. This places a high premium on adaptability.

To a degree one can identify a hierarchy of stability. The science that underlies the design of new types of devices and the use of new materials is updated the least frequently. New processing techniques and equipment are developed more frequently, but not as frequently as are new manufacturing plans, and test-structures. This is one reason that most applications of symbolic, knowledge-based reasoning have preferred to concentrate on individual processing operations. It is difficult to represent knowledge about complete manufacturing processes in an manner that enables it to adapt to changes in the manufacturing plan. This is a primary focus of our research.

# Chapter 3. Experiential Knowledge

In this chapter we briefly describe experiential knowledge, and introduce heuristic rules as a method of representing and reasoning with experiential knowledge. We then describe the concept of a causal association network, as a variant of the heuristic-rule paradigm that is particularly appropriate for diagnostic tasks in engineering domains. Next we describe the particular representational machinery we employ: the HyperPIES system [10, 11]. Finally, we discuss the relative advantages and disadvantages of this approach to knowledge representation in the context of diagnosis in the semiconductor manufacturing domain.

## 3.1. What is Experiential Knowledge?

In this document we distinguish between experiential and theoretical knowledge as though they were two categorically separate kinds of things. In truth, these two types of knowledge are at best fuzzy categories. There is a spectrum in the degree to which knowledge is based directly on experience, as opposed to being driven by aesthetic universal principles. The content and the source of the knowledge determine whether it should be deemed experiential or theoretical, and what advantages and limitations it has. However, in practice content and form go hand in hand in symbolic representations of knowledge, because of the way the knowledge is acquired and how it is used. Even when the source of knowledge might be deemed theoretical, if it is represented in the fashion usually reserved for experiential knowledge, it often inherits the same limitations. Thus, our working definition of experiential knowledge necessarily confuses content and form to a degree. For our purposes, experiential knowledge has several traits:

a) Experiential knowledge consists of a relatively large collection of facts;

b) These facts usually concern quite specific situations, identified by a set of characteristic properties, so that each fact has a limited range of applicability;

c) The facts make predictions about additional properties that should be expected of the situation: for this reason, the facts represent potential inferences;

d) The facts rarely invoke universal principles to articulate a rationale for belief in the expected additional properties;

e) There is often uncertainty about the predictions, either because the situations are not completely characterized (so that new situations that match the stated characteristics are not truly sufficiently similar ), or there is a source of noise or randomness; and

f) The potential inferences that the facts represent tend to be "large." That is, if one could imagine explaining the rationale for the inference in more theoretical terms, the explanation would be a long one.

Experiential knowledge is most often obtained through a process called "knowledge acquisition," and represents a codification of the experience of one or more human experts in a specific domain. The desire for compactness and efficiency, and the tedium involved in acquiring the knowledge both contribute to the tendency for the individual inferences to be large. An example of a simple but large inference from everyday experience is: in a car with a dead battery, turning the key does not start the engine. Explaining this inference requires explaining the relationships between the battery, the key and parts of the engine, such as the starter motor, the ignition system, the spark plugs and the alternator.

## 3.2. Representing Experiential Knowledge

Heuristic rules [12] are the primary technology employed to represent experiential knowledge. A rule is a conditional statement of the form "IF <*situation*> THEN <*action*>." Such rules directly associate observable conditions described in the <*situation*> component of the rule with conclusions that can be drawn or actions that should be taken described in the <*action*> component of the rule. They are called "heuristic" because there is no guarantee that they are correct in all contexts: the reasoning system assumes that the rules are independent of one another, except to the degree that they "chain." Rules chain when the conclusions drawn by one rule affect the situation recognized by another.

An "inference engine" employs the rules, which are collected into a (rule-based) "knowledge base," to draw inferences in one of two ways. A forward-chaining inference engine works by matching the <*situation*> components of the rules to its representation of the state of the world. It then draws the conclusions warranted by the <*action*> components of the rules for which the matching process succeeds. A backward-chaining inference engine matches the <*action*> component of the rules against the world-state, and hypothesizes that the <*situation*> descriptions of the rules that match are valid. The

system may represent the state of the world using sentences in a simple declarative language (such as object-attribute-value triples), or it may employ "frame-based" representations that represent objects using descriptive "slots" and provide a classification system with inheritance.

In many cases, the rules in diagnostic rule-based systems represent associations between observable manifestations of failures, and the events that caused them. The rules are often written in the causal direction, that is, they take the form "IF <*fault occurs*> then <*manifestation will be observed*>." Then a backward-chaining inference engine uses the rules to determine what faults might explain those manifestations that have been observed. When the knowledge base takes this form, the rules constitute a partial causal model of the behaviour of the system being diagnosed. It is partial because it only addresses causal connections between abnormal behaviours: it does not explicitly model the mechanisms involved in the normal behaviour. Further, it can only weakly address interactions between faults, as the causal associations are assumed to be independent.

Some researchers have recognized the "partial causal model" nature of such knowledge bases and made it explicit, representing diagnostic knowledge as an explicit network of causal associations. Structuring the knowledge base this way facilitates knowledge acquisition. The first system to use this approach, in the domain of medical diagnosis, was called CASNET (Causal ASsociation NETwork) [13]. In this system the causal network made it possible to capture the time-behaviour of progressive diseases.

The causal association network representation enables a hybrid of forward- and backward-chaining reasoning highly suited to the diagnostic task. Following associations backwards from manifestations to the faults that can explain them can be used to generate *hypotheses*. The hypotheses can then be *verified* by following the associations in the forward direction to determine whether additional manifestations expected of the hypothesized faults are actually present.

## 3.3. HyperPIES[1]

The first system that employed an explicit causal network for diagnosis in the semiconductor manufacturing domain is described in [10]. This system, called PIES™ (Parametric Interpretation Expert System) was developed to perform the parametric interpretation task. This diagnostic task consists of examining the results of parametric measurements on the devices and special test structures manufactured, and determining, for those wafers with measurements that fall outside acceptable range, what might have caused the failure.

The system is designed to facilitate knowledge acquisition by domain experts. It imposes a structure that is specific to the manufacturing domain, and organizes the knowledge as a set of cases. The imposed structure consists of four levels at which abnormalities can be described: Measurement-Deviation, Physical-Anomaly (wafer structure), Process-Anomaly, and Root-Fault. Within each of these four main categories of classification, the abnormalities can be further classified. For example, Root-Fault abnormalities are further classified into four categories—equipment failure, human error, bad source materials, and bad environment; Process-Anomaly abnormalities are classified by type of process involved and type of deviation; Physical-Anomaly abnormalities are classified by the type of structure involved; and Measurement-Deviation abnormalities are classified by type of measurement and the structure on which the measurement is made.

All causal associations are between nodes at one of these main levels (intralevel associations) or between two adjacent levels (interlevel associations). The latter make up the majority of the associations captured. For example, an abnormal structure on the wafer is invoked to explain an abnormal measurement, by having an interlevel causal association “link” in the network between a node at the Physical-Anomaly level and a node at the Measurement-Deviation level. Abnormalities in processing, represented by nodes at the Process-Anomaly level, are linked as possible causes of abnormal wafer structures. Finally, Root-Fault nodes represent ultimate causes for manufacturing failures, and are causally linked to nodes at the Process-Anomaly level. When viewing a

---

[1] PIES, HyperPIES, HyperClass, MetaClass and Class are trademarks of Schlumberger Technologies, Inc.

node in the network, one can immediately and easily see what nodes it is associated with, and thus what the possible causes are for the abnormality it represents, and what other abnormalities it potentially explains.

The PIES system was originally written in Common LISP with a character-based interface intended to work on “dumb” terminals. Since then, the program has been ported to the HyperClass™ environment developed by Schlumberger, Inc. This environment extends Common Lisp with an object-oriented programming system called Class™ and a set of objects, collectively called MetaClass™, that facilitate creating graphical user interfaces (GUI’s). The net impact of this port is to provide PIES with a mouse and windows graphical user interface. The ported system is called HyperPIES™. This is the system that will be used to represent experiential knowledge in this work.

In HyperPIES™ the abnormalities are represented by a class of programming objects called *Fault Concepts*, which are organized into a taxonomic hierarchy with the four main levels as the first level of differentiation. For abnormalities that represent deviations in a continuous physical quantity, the system requires separate fault concepts for each of several qualitatively distinct classes of failure, indicated by the terms *very-low*, *low*, *high*, and *very-high* (for example, IDSAT-VERY-LOW).

Causal associations among the fault concepts are represented by a class of programming objects called *Causal Links*. Uncertainty is handled by a measure called *association-strength*, that may take on one of several qualitative values: *must*, *very-likely*, *likely*, *probably*, and *maybe*. As the value names suggest, association-strength is a probabilistic concept. However, it is entirely subjective, and the space of events over which these probability densities might range is left undefined.

Figure 3.1 illustrates the top-level “control panel” editor that the user is presented with upon starting the PIES system. It provides five top-level functions, which are quite self-explanatory. Invoking the “INSPECT existing fault concept” function by mouse-clicking on either the name/explanation text at the top of the editor or on the leftmost of the “buttons” that span the central area of the editor causes a simple menu-like editor to appear as at the left in figure 3.2 below.

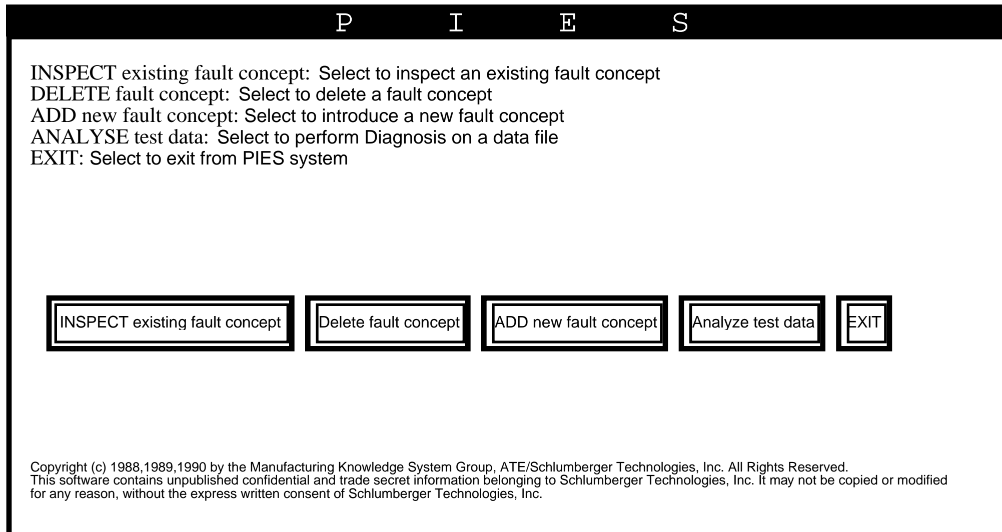


Figure 3.1 HyperPIES top level control panel

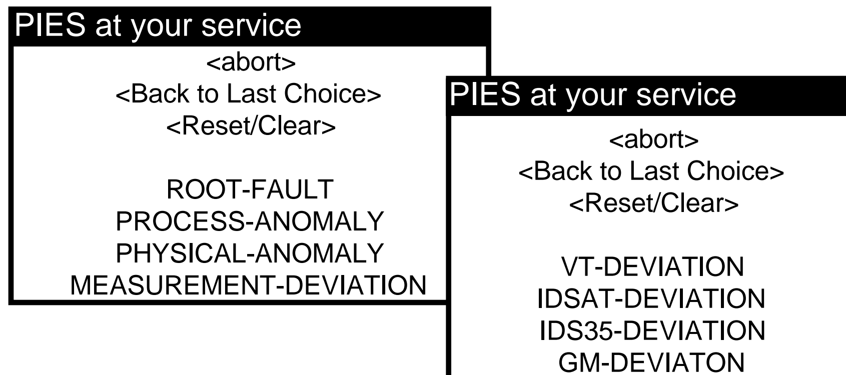


Figure 3.2 Traversing Fault Concept Taxonomic Hierarchy

The first version of the menu indicates the top-level differentiation of fault concepts into the four main abnormality levels. Selecting any one of these with a middle mouse click causes the menu to disappear and reappear showing the children of the fault concept class that was selected. In the figure, selecting MEASUREMENT-DEVIATION results in a new menu showing the four classes of measurement abnormalities captured in the knowledge base. Selecting a fault class with the left mouse-button brings up a fault-concept editor that can be used to indicate causal associations between the selected fault concept and others selected through a similar process. However, the system intends that causal associations be established only among fault concepts at the leaves of the fault-concept classification tree. If a user left-clicks on a non-leaf fault-concept class, the system warns the user with a dialog box that permits the user to change his/her mind or go ahead.

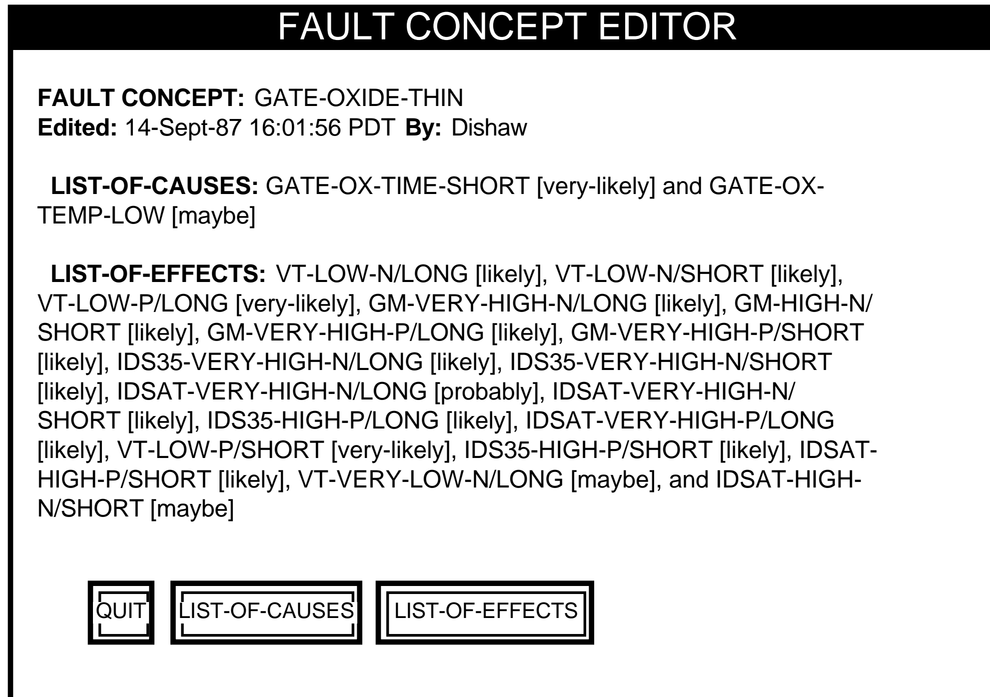

Figure 3.3 Fault Concept Editor

Figure 3.3 displays the fault concept editor for the fault concept GATE-OXIDE-THIN. The two main fields displayed are the LIST-OF-CAUSES, which lists the causal associations between this fault concept and those that might explain it, and the LIST-OF-EFFECTS, which lists the causal associations between this fault concept and those that it might explain. The buttons containing the names of these fields can be used to invoke a menu that allows for the inspection, addition, or deletion of causal associations. The inspection and addition options invoke the CAUSAL-LINK-EDITOR, shown below. Clicking on the bold field names with the middle mouse button enables the user to edit the contents of that field. Some fields, such as the CAUSE-LEVEL, EFFECT-LEVEL and LINK-TYPE fields, are automatically determined by the user's choices for other fields.

Diagnostic reasoning in PIES follows a multilevel hybrid approach similar to that described in the previous section. The parametric measurements are first processed to abstract qualitative symbolic symptoms from the quantitative data. Statistical techniques are employed to remove invalid "outlying" data points considered to have arisen from bad electrical contacts. Then mean and standard deviation statistics for each measurement are computed for all the wafers in the lot being diagnosed. Finally, these are compared with limits provided by process engineers to classify measurements as

very low, low, normal, high, or very high. These classifications constitute the initial set of measurement-level symptoms.

**CAUSAL LINK EDITOR**

**CAUSAL LINK (name for reference only):** GENERIC-CAUSAL-LINK-98
**Edited:** 10-Nov-87 13:58:04 PDT **By:** Dishaw
**CAUSE:** GATE-OX-THIN
**CAUSE-LEVEL:** PHYSICAL-STRUCTURE-LEVEL
**EFFECT:** VT-LOW-P/SHORT
**EFFECT-LEVEL:** MEASUREMENT-LEVEL
**LINK-TYPE:** INTER-LEVEL-CAUSALITY
**ASSOCIATION-STRENGTH:** VERY-LIKELY
**OBSERVATION-COUNT(^):**
**REFERENCE-LIST(^):** <...empty...>
**COMMENT-LIST(^):** <...empty...>

QUIT

Figure 3.4 Causal Link Editor

The diagnostic procedure progresses level-by-level through the four levels of abnormalities, repeating for each level the same hypothesis and verification cycle. The causal links are followed backwards from the initial set of symptoms (or the hypotheses at the previous level) to form a set of hypotheses at the next higher level. This set of hypotheses is augmented by abnormalities at the same level linked through intralevel causal associations. The causal links emanating from these hypotheses are then followed in the forward direction to form expectations regarding additional symptoms at the next lower level. These expectations are compared with the failure hypotheses at that level that have been concluded thus far. These comparisons result in matching scores for each failure hypothesis, indicating how well the hypothesis' expected symptoms match the current situation. The hypotheses are sorted by their scores, and hypotheses with sufficiently low scores are eliminated. The remaining hypotheses form the symptom set for the next stage of reasoning. In this way, Measurement-Deviation abnormalities are used to generate and verify hypotheses at the Physical-Anomaly level, Physical-Anomaly abnormalities generate and verify hypotheses at the Process-Anomaly level, and Process-Anomaly abnormalities generate and verify hypotheses at the Root-Fault level.

The MetaClass™ system includes methods for generating displays of node-and-arc graphs. This facility is primarily used to display the directed acyclic graphs induced by the taxonomic relationships among classes. However it can also be invoked to display other graphs. The HyperPIES system uses it to display the results of its diagnostic reasoning as a graph. The leftmost nodes in the display are the hypothesized diagnoses in order of likelihood from top to bottom. From these arcs extend to the nodes that represent evidence for the hypotheses. This analysis-results graph is thus a sub-graph of the complete causal association graph. Figure 3.5 illustrates a simple version of such a causal analysis graph, for a case in which a single Process-Anomaly fault is concluded.

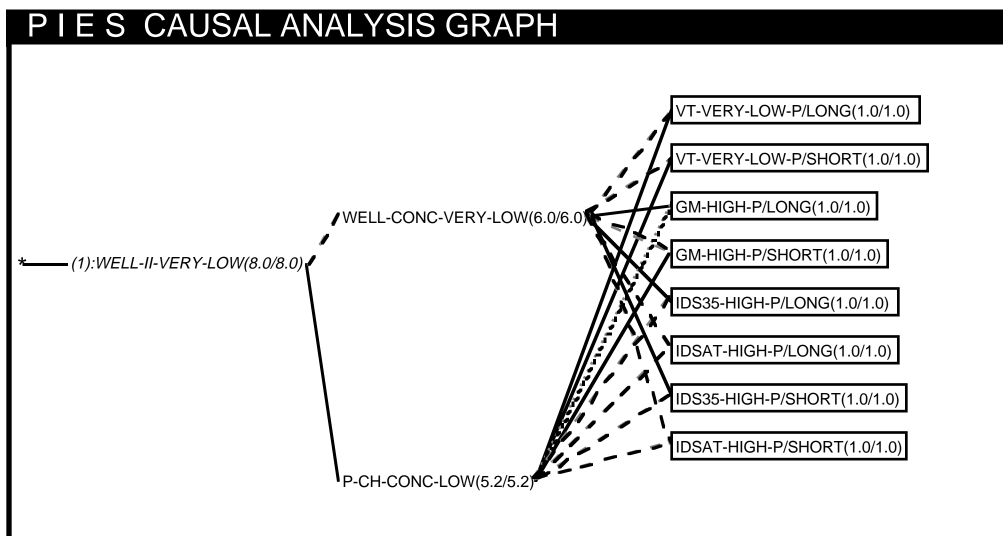


Figure 3.5 PIES Causal Analysis Graph

## 3.4. Strengths and Weaknesses of Experiential Knowledge

In a causal association network, only one predicate is being manipulated by the computer. This predicate might be called CAUSES (as in A CAUSES B). A link in the network indicates that the event or condition represented by one node is a potential *cause* for the event or condition at the node it is linked to. Inversely, the event or condition represented by the second node is a possible consequence or manifestation of the event or condition represented by the first node. The nodes being linked, however, are atomic and opaque: they can represent anything. Indeed, a node consists of little else than a symbolic name for the condition or event it stands for.

This representational scheme provides great flexibility. One can indicate potential causal relationships between any two phenomena that one can give names to. However, this flexibility comes at a cost. There are two interrelated problems connected

with it that together are often described as “brittleness” of the knowledge base. The term brittleness refers to the inability of knowledge-based systems to adapt to changes in context. The semiconductor world is characterized by rapid and continual change. Semiconductor manufacturing plans have a short life-span. Thus, it is a serious shortcoming if a system designed to diagnose a manufacturing plan cannot adapt quickly to changes in the plan.

The first problem is that the applicability of the knowledge in an experiential knowledge base depends on contextual assumptions that are not explicitly represented. In our case, these contextual assumptions concern the structure of the manufacturing plan. As a consequence, such knowledge might be applied inappropriately in a new context. The second problem is that the applicability of the knowledge is unduly restricted by the specificity of the terms employed (the choice of symbolic names used to represent events and conditions). This can lead to the possibility that knowledge that is relevant might not be applied.

These two problems are present to some degree in all knowledge-based systems, regardless of the knowledge-representation used. However, systems based on direct associations between high-level concepts are especially prone to these problems. In particular, the contextual assumptions are not made explicit because they cannot be usefully represented: the impact of a change in these assumptions cannot be deduced, as the knowledge is not represented with sufficient detail to permit an indication of how the associations depend on the assumptions. Similarly, knowledge encoders use overly specific terms because they are unable to describe in a general way how classes of concepts are interrelated. In both cases, alleviating the problem requires representing knowledge of the domain with greater detail, at a finer level of “granularity.”

Consider this example of an association from the PIES knowledge base (entitled AESOP) constructed by Patrick Dishaw [11]:

POLY-GATE-UNDER-ETCH →CHANNEL-LENGTH-LONG

This association reports a connection between a processing anomaly (under etching of the polysilicon gate) and a physical structure anomaly (overly long channel length). The association was determined by statistical experiments with a simulator. However, much is understood concerning why such a connection should occur.

At one stage of processing, a layer of polysilicon sits over a thin layer of silicon dioxide, which in turn sits over the single crystal silicon substrate. The polysilicon layer has a layer of photo-sensitive resist above it which has been “patterned” (i.e., selectively removed in some areas but not others) by a sequence of photolithography steps (figure 3.6).

An Etch step is then performed, during which the polysilicon in those areas not covered by resist reacts with a plasma of a reagent, and is thus removed little by little, working from the exposed surface towards the interface with the silicon dioxide. The result is that the polysilicon is completely removed from the wafer in those areas not covered by resist. The pattern in the resist is thus “transferred” into the polysilicon layer.

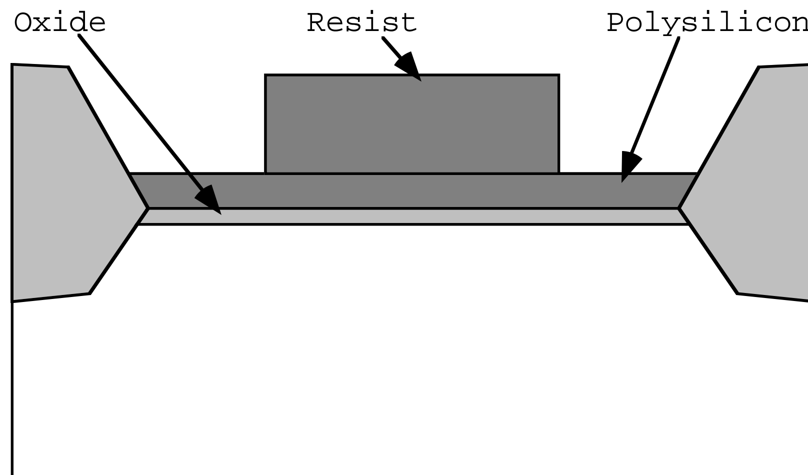

Figure 3.6 Configuration just before Poly Gate Etch

An electric field directed perpendicular to the surface of the wafer helps to make this etching process anisotropic: the process works faster in the “vertical” direction than in the “horizontal” direction. Nonetheless, some material is removed laterally as well as vertically, so that some material is removed from under the resist, and the “sidewalls” of the remaining polysilicon are at a slant (about 70 degrees, in this case; figure 3.7).

Indeed, the etching process is purposely continued about twenty percent longer than is required to etch through the vertical thickness of the polysilicon layer so that this lateral etching can move the lateral boundaries of the polysilicon under the boundaries of the covering resist.

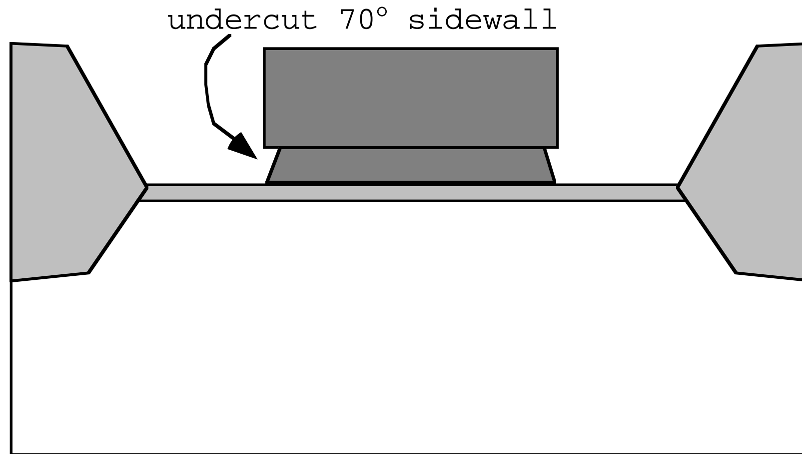


Figure 3.7 Configuration after Poly Gate Etch

Later in the process an Ion Implantation step is performed. During this step, ions of a dopant are directed at the wafer with sufficient energy that they penetrate the surface and come to rest at some distance below the surface. In those areas not covered by polysilicon, the ions enter the silicon substrate, whereas in the covered areas the ions come to rest primarily in the polysilicon layer.

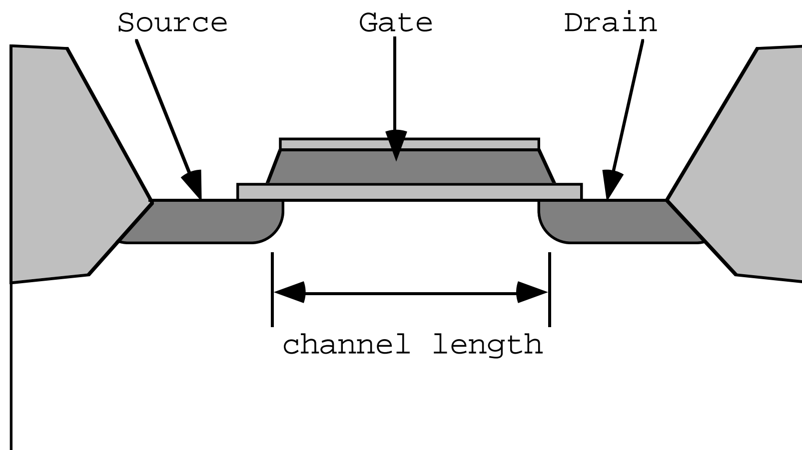


Figure 3.8 Formation of MOSFET

Still later an Annealing step will cause the ions to be incorporated into the crystal lattice of the silicon, forming p- or n-type doped regions, depending on the type of the dopant. Thus, the inverse of the pattern of polysilicon is transferred into a pattern of p- or n-type doped regions in the silicon substrate (figure 3.8).

The resulting structure is a Field Effect Transistor. The polysilicon serves as the GATE of the transistor, and the doped regions form the transistor's SOURCE and

DRAIN terminals. The CHANNEL LENGTH is the lateral distance between the edges of the source and drain regions, i.e., the length of the region between them that lacks dopant ions.

In the diagnostic association, the process-level anomaly named POLY-GATE-UNDER-ETCH refers to a failure condition in which the ETCH step that defines the POLYsilicon GATE is performed for too short a period time. The expected result is that the lateral etching described above takes place for a shorter time, and thus the gate structure is longer than it should be. Since the polysilicon of the gate acts as a mask to the control the implantation of dopant ions in the Ion Implantation step, the increased length of the gate is ultimately reflected as an increased CHANNEL LENGTH.

The context not represented in this case includes the following:

1) the process used by the Etch step in question is not completely anisotropic (it is possible for the degree of anisotropy to be much higher, so that very little lateral etching takes place at all; in this case a more isotropic process was employed because the sloping walls that result make laying metal wires over the structure less troublesome);
2) the etch operation is masked: in some areas the wafer is covered with a material (photoresist) that resists etching, while in others the polysilicon is exposed to the etch process, so that etching occurs only in the exposed areas;
3) the duration of the etch step is intentionally extended to compensate for the sloping walls by moving them back under the resist;
4) the process is “self-aligned”: that is, the edges of the source and drain regions are determined directly by the edges of the gate, rather than by photolithography with another mask.

At the same time, the association is too specific. Clearly, any time that lateral etching is taking place a failure to etch sufficiently will have an impact on the lateral positions of the walls formed by the step. However, this association applies only when polysilicon is being etched, and only refers to the impact on a single type of structure (the MOSFET Gate), despite the fact that the very same etch step is also simultaneously creating other structures, such as polysilicon wires and capacitors. Although additional causal associations might link the over-etching of the polysilicon gate with physical anomalies in other structures being simultaneously manufactured, there would be no representation of the fact that these additional associations are instances of the same general phenomenon. Each such association would have to be added independently. This fact contributes to potential problems with the completeness of an experiential

knowledge base. In truth, the AESOP knowledge base does not have these additional associations, because the associations were obtained by simulating the manufacturing plan using numerical simulators, and these other structures were not among those whose manufacture was simulated. (Any model-based knowledge base would suffer the same incompleteness when the structures reasoned about do not include all typical structures being manufactured.)

Further, whenever an Ion Implantation step is masked by a structure on the wafer, any deviation in the lateral positions of that masking structure's edges will affect the edges of the regions formed by the implanted dopants, but this association is restricted to talking about the channel length of MOSFETs.

In part, the over-specificity in this example was an effort to incorporate contextual information. There are several etch steps in any process, and there are many structures created on the wafer, but it is necessary to identify the etch steps and structures that correspond to each other: which etch steps define the edges of which structures? Which etch steps and structures are appropriately CAUSALLY linked? In the example, this question is finessed by referring to a specific etch step and a specific structure known to be causally linked. This solution is necessary because the PIES system has no representation of the manufacturing sequence, the wafer structures created, or how these relate to one another in general.

# Chapter 4. Theoretical Knowledge

In this chapter we briefly describe model-based reasoning as a method of representing and reasoning with theoretical knowledge. We then discuss some aspects of the manufacturing domain in general, and the semiconductor manufacturing domain in particular, that introduce additional complexity to the model-based reasoning approach. Next we describe the general character of the modeling technology we employ: the Gordius system [14], comparing and contrasting it with first-order predicate logic and calculus. After that introduction, we continue with a detailed discussion of the elements of the representational machinery. Then we discuss in some detail how the machinery is used to represent theoretical knowledge of the semiconductor manufacturing domain. Finally, we summarize some of the main points of the chapter.

## 4.1. Model-Based Reasoning

One alternative to driving the diagnostic task with experiential knowledge is to appeal to theoretical knowledge. In the case of engineered systems, the relevant theoretical knowledge available concerns how the system works, or why it works the way it does. The key idea is that understanding how the system works is an aid, though not a prerequisite, to diagnosing why the system might be failing.

The term “Model-Based Reasoning” refers to reasoning about the behaviour of systems using models of the mechanisms underlying that behaviour. The design of these models supports generation of causal explanations for the behaviour of the system. In some cases, causal descriptions are obtained from acausal (equation) models via a process call *causal analysis* [15]. One can distinguish two approaches to model-based reasoning. The first approach, which we term “holistic,” directly describes the behaviour of the entire system [16]. This description is composed of functional dependencies and behavioural events. The behavioural description may or may not reflect the structure of the system. The system’s behaviour is described directly, not inferred from the system’s structure. The second approach is “reductionist”: the system is decomposed into sub-components which have behavioural models [3, 17-20]. The behaviour of the system as a

whole is then inferred from the models of the behaviour of the sub-components and the description of how the sub-components are connected to form the system (the system's structural description).

Model-based techniques can represent knowledge at a greater level of detail than is generally possible with direct causal associations or heuristic rules, without a concomitant increase in the size of the knowledge base required. This is because the models they employ are compact axiomatic systems from which large amounts of detailed information can be derived by deduction. This is especially true for the reductionist approach, wherein modeling the behaviour of a few components permits the generation of behavioural descriptions for large complex systems composed from those components.

Diagnosis is performed by propagating information about observed anomalies backwards along the "causal pathways" of the system to identify what components of the system might be responsible for the observations. The causal explanation description of the system's behaviour generated by model-based reasoning identifies these causal pathways.

In addition to the small size of the knowledge base that one has to construct, the key advantage of the model-based approach stems from the generality of the models, which leads to a form of robustness. The models are written in a sufficiently general fashion that they apply to a wide variety of potential situations. In the case of the holistic approach, the models are written to correctly predict the response of the system to a wide variety of possible stimuli. In the reductionist approach, the component models are written to correctly predict the behaviour of the components (or correctly predict what processes will act) in a wide variety of contexts in which the components may be used. This in turn implies that the behaviour of a wide variety of systems composed of these components can be predicted, and explained in causal terms.

The reductionist approach can be further subdivided into two categories: device-centered and process-centered. The device-centered approach attaches a model of behaviour directly to each sub-component of the system. It is called device-centered because the construction of models is centered on describing how these sub-components, or *devices* behave. A prime example of the device-centered approach to reductionist model-based reasoning is the work on reasoning about digital circuits [20,

21]. In this work, the behaviours of individual gates and/or sub-assemblies such as registers are described, and the behaviour of a complex circuit composed of such components is inferred.

Rather than attaching behaviours to devices, the process-centered approach concentrates on describing the world in terms of continuous phenomena called *processes* that spontaneously occur whenever specific circumstances hold [18]. These processes act like Newtonian forces, influencing particular physical variables to change continuously. Like forces, when several processes influence the same physical variables, their impact is determined by summing these influences. This approach is most successful when applied to systems describable using first-order ordinary differential equations.

## 4.2. Special Considerations of the Manufacturing Domain

In our domain, the 'system' to be reasoned about is the manufacturing process, which is driven by a manufacturing plan. The constituent components of the process are the individual manufacturing operations called for in the plan. In manufacturing domains which involve assembly of sub-assemblies, the topology of the system is a tree that branches backwards in time: the assembly operations are the nodes with multiple branches, because they bring together independently manufactured sub-assemblies. In the semiconductor domain, the topology of the system's structure is quite simple: the operations are performed in linear sequence (we shall not be concerned, for the most part, with the minor ways in which real semiconductor manufacturing processes deviate from a linear sequence).

A device-centered reductionist approach is most appropriate for the semiconductor manufacturing domain. In this domain, the 'devices' to be modeled are the manufacturing operations. The models describe the behaviour of each operation by describing the impact of each operation on the materials being processed. In this way the diagnostic system can adapt to changes in the manufacturing plan, because the manufacturing plan is an explicit input into this computational process. The causal pathways through the manufacturing process are inferred by reasoning with models of the individual operations.

There are different levels of granularity at which the manufacturing operations might be modeled. At one level, each operation can be viewed as a discrete action which

has a well-defined impact on the product being manufactured. At another level, the modeler can recognize that a single operation can span an interval of time during which processes act continuously to modify attributes of the product being manufactured. Our focus is on the manufacturing process as a whole. We will be concerned with how the individual operations interact and combine to form a complete manufacturing process. Thus, we will concentrate on the discrete action level of granularity. Indeed, for that reason we call the model-based component of our reasoning system the Discrete Action System (DAS). A system that focused on the diagnosis and/or control of a single operation might prefer to reason at the continuous process level of granularity.

The manufacturing domain differs in one important respect from most domains in which device-centered model-based reasoning has been applied previously. This technique has been applied in domains in which the topology of the system's structure is fixed, and the signals that are transferred between the system's components are relatively simple. For example, in the digital circuit domain the structure of the system is given by the topology of interconnections among the gates, and the signals that pass between the gates are simple boolean values.

There are two ways to view the situation in the manufacturing domain. One way is to consider that the components of the system are the operations, and that the topology of the system's structure is the fixed topology of the manufacturing plan. In this view, the signals that pass between the components are descriptions of the products being manufactured. These signals are quite complex. The description of the product is highly structured, and does not necessarily have a fixed number of variables. For example, in the semiconductor domain the product is the wafer and its description includes many variables for the attributes of an *a priori* undetermined number of regions and physical layers. The number of variables and the structure of the signal is determined by the behaviour of the manufacturing process — what kind of product it produces.

On the other hand, one could decide that the signals are the variables representing individual attributes, and the topology of the system is the topology of functional dependencies among these variables. In that case, the system's topology is not known *a priori.* One must simulate the manufacturing process to determine what the variables are and what the topology of the functional dependencies is among them.

Further, when diagnosing failures, one must contend with the possibility that the mode of failure induces a change in the topology of the functional dependencies.

Regardless of which view one prefers to take, the key issue is that before diagnostic reasoning can take place it is necessary to assimilate the manufacturing plan by symbolically simulating it to determine the structure of the signal and/or the topology of the system. If fault modes that change the topology of the wafer are to be handled, then it is also necessary to simulate such faults.

This need for assimilation is also true of systems that employ the process-centered approach to model-based reasoning. It is necessary to perform what is called an “envisionment” or at least a simulation to determine what processes might be acting, and thus what functional dependencies exist in the system. An envisionment attempts to create a finite discrete description of all states that the system might enter, and all ways that the system might make a transition from one state to another. A simulation starts from a single initial state and traces a single path or a forward-branching tree of paths through states attainable from that initial state.

It is sometimes the case that one cannot directly observe characteristics of the product manufactured. This is indeed true in the semiconductor domain. In such cases, the failures to be diagnosed are detected as anomalies in the behaviour of the product. Thus, as well as modeling and reasoning about the behaviour of the manufacturing process, to perform diagnosis one must also model and reason about the behaviour of the product that was manufactured. In this work we model the behaviour of the devices and test-structures created on the wafer with simple analytical expressions. These expressions describe the behavioural features of devices as functions of their physical features. The efficacy of this approach was demonstrated in [6, 7] and in [4, 5].

## 4.3. The Modeling Technology

### 4.3.1. The Gordius System

The Discrete Action System is built on a system called Gordius [14] that was designed for reasoning qualitatively about plans. This subsection describes that system, comparing it with Predicate Logic and the Predicate Calculus.

#### 4.3.1.1 Gordius versus Predicate Logic

The key input to the Gordius system is a description of the actions of which plans are composed. This description includes statements about the preconditions necessary for execution of an action, the effects of executing an action, and any constraints that might hold while the action is executed. The actions can be parameterized, in which case the description names these parameters and indicates their types.

These statements are made in a formal language that consists of a Lisp syntax for First-Order (Predicate) Logic (FOL) [22]. However there are several distinctions to be made between the Gordius language and predicate logic. Predicate logic is a system for expressing facts. The logic is concerned with the syntax for expressing facts, how meaning is assigned to the resulting expressions, and what other facts are *entailed* by a given set of facts. The predicate calculus is concerned with rules of inference: how new facts are derived from old ones. This can also be viewed as: how the entailment of new facts by old ones is proven. The Gordius language is an extension of the predicate logic, but only a subset of the predicate calculus.

The first extension is that the Gordius language is a typed logic. All constants, functions and variables in the language have a specific type, and quantified statements are always quantified over individuals of a specific type or elements of a set that is a subset of all individuals of a specific type. The language employs an object-oriented approach to defining types and specifying for each object-type what attribute functions apply to it. A simple form of taxonomic hierarchy is supported, and information about the applicability of attribute functions is inherited.

The second extension is that the Gordius language includes a simple form of nonmonotonicity. Reasoning about plans requires reasoning about time. Gordius reasons about time explicitly. That is, time is an explicit variable in expressions concerning the value of an object's attributes and in statements about change. Any system that explicitly indexes the values of functions or the truth of statements by time incurs a problem known as the *Frame* problem [23]. Stated simply, the frame problem arises because we cannot deduce anything about the world at a given time from knowledge of the world at a previous time, unless we somehow axiomatize inertia — the fact that truths tend to persist through time. Gordius handles this problem by making *persistence assumptions.* Whenever an object is created or an attribute of an object

is changed to a new value, Gordius explicitly assumes that the new object continues to exist or the attribute retains its new value forever, unless it can prove otherwise.

Here is where nonmonotonicity comes in. Normally, in predicate logic the addition of new facts to a knowledge base does not change the set of facts entailed by those facts previously in the knowledge base. One says that the body of facts entailed grows monotonically as new facts are added. In Gordius, the addition of new facts can change the set of facts entailed, because the new facts can contradict the persistence assumptions. For example, the discovery that an etching operation removes a layer from the wafer contradicts the assumption that the layer continues to exist forever. Thus, the Gordius system is nonmonotonic: facts previously held to be true, such as that a layer's existence persists beyond a certain time, can become false as new information is obtained. This approach is similar to that of Yoav Shoham's *potential histories* [24]. The situation is simplified in Gordius by the fact Gordius does not have to reason about the persistence of continuous change such as motion. All change is confined to occur within the time intervals spanned by actions. Between these dynamic intervals, all values are quiescent, and Gordius only reasons about the persistence of these quiescent values. The fine time-structure of events internal to a dynamic interval is not reasoned about: only the ultimate impact of the action is described. Thus, Gordius need not concern itself with the problem of deciding when potential histories intersect.

#### 4.3.1.2 Gordius versus Predicate Calculus

The reasoning mechanisms of Gordius differ from predicate calculus in two significant ways. The first way concerns concrete versus abstract reasoning, and the second concerns semantic attachment. Predicate calculus is said to be both *sound* and *complete.* The soundness of predicate calculus implies that only facts that are entailed can be deduced. The completeness of predicate calculus implies that every fact that is entailed can be deduced. Unfortunately, predicate calculus is also only semi-decidable. If a fact is not entailed, then the predicate calculus inference procedure is not guaranteed to terminate. This means that it can take an infinite amount of time to decide whether a given fact is deducible. Further, even when facts can be derived in a finite amount of time, the amount of time it takes may be an exponential function of the number of facts in the knowledge base. The two major ways that Gordius differs from predicate calculus are both aimed at avoiding this time-complexity. Unfortunately, one difference (concrete reasoning) makes Gordius incomplete, and the other (semantic attachment) makes the soundness of Gordius difficult to demonstrate and maintain.

To describe the first difference, we introduce a distinction between *abstract* and *concrete* reasoning. By abstract reasoning, we mean the inference of new general statements from other general statements. Consider the two statements: (*a*) "Every mammal is either male or female, but not both," and (*b*) "If a mammal can bear children, that mammal is female." These are general statements about all mammals. From these two statements, we can easily infer that (*c*) "If a mammal can bear children, that mammal is not male." This is also a general statement about all mammals. The inference of (*c*) from (*a*) and (*b*) is an abstract inference: no reference to specific individuals was required to make it. This sort of inference is within the purview of predicate calculus, though the actual derivation might be longer than one would hope.

Abstract inference of this sort is not possible within Gordius. Gordius can only perform concrete reasoning, which we define as reasoning about specific individuals. Gordius accepts general statements of the form (*a*) and (*b*), but it can only apply them to specific individuals. Thus, for every specific child-bearing individual Gordius will be able to conclude that the individual is female and therefore not male, but it will not be able to explicitly conclude the general statement that all child-bearing individuals are not male. Note that this limitation does not in any way affect the meaning of the statements made — it only affects the ability of the system to deduce the consequences of these statements. In particular, the system will not deduce any general consequences.

While this may seem like a very severe limitation, it does not turn out to be very important for our application, as we are primarily interested in how the general statements about the effects of actions affect specific individuals (for example, specific layers and regions on wafers). Also, it is a limitation that buys tremendous efficiency, because it reduces reasoning in predicate logic to the complexity of reasoning in propositional logic, which is completely decidable (though still exponential).

This limitation to concrete reasoning is due to the treatment of quantified statements. General statements are made in predicate logic by introducing variables into the expressions where terms representing individuals are appropriate, and *quantifying* those variables. There are two types of quantification. *Universal* quantification is used to express statements that apply to every member of a class of individuals. The statements *a* and *b* in the example above are both universally quantified. We can rephrase the English to better reflect how they would be written in predicate calculus: (*a*) "For every x that is mammal, x is female and x is not male, or x is male and x is not female," (*b*) "For

every x that is mammal, if x can bear children then x is female.” Statement *c* above is also a universally quantified statement. Gordius handles universally quantified statements by doing two things: it applies the statement to all appropriate individuals that it knows about and it sets up what is called a *daemon*. A daemon is a rule that fires whenever the system determines that a new individual (of appropriate type) exists. The daemon applies the quantified statement to every new individual that comes along. Applying the quantified statement to an individual means substituting the constant naming the individual for every occurrence of the quantified variable in the statement. This turns the quantified statement into a ‘ground’ statement, which has no variables (and no quantifiers). Ground statements can then be treated as propositions.

*Existential* quantification is used to express statements about the existence of individuals with certain characteristics without naming the individuals. The statement “John has a mother” employs existential quantification. This statement would be expressed: “There exists x that is human, such that x is the mother of John.” For concrete reasoners, existential quantification is problematic precisely because no specific individuals are identified. For this reason, existentially quantified statements should not be used as assertions in Gordius. They can be used, however, in places where they have a negative sense. This is because the negative of an existential statement is a universal statement, and vice versa. Thus, one can use an existentially quantified statement in the ‘if’ part of an ‘if-then’ implication, because the ‘if’ part has a negative sense. It is easy to test whether it is known that, “John has a mother.” For the same reason, one should not use universally quantified statements in the test part of an implication (or other place where it would have a negative sense). The system can never be sure whether all individuals of a given type satisfy a condition, because it can never know that it has met all individuals of that type. It also means that neither type of quantified statement can be used on either side of an equivalence statement. Both sides of an equivalence statement have both positive and negative senses, as an equivalence statement such as “A if and only if B” is logically equivalent to “if A then B and if B then A.” The exception to this rule is universal quantification over the members of a finite closed set, since in this case the system knows all the individuals involved (and knows that it knows!).

There is one exception to this rule regarding the use of existential quantification in Gordius. Gordius provides a quantifier called “`CREATED`,” which can be used to indicate that the effect of an action introduces a new individual of a given type at a specific time. This is essentially an existential quantifier, equivalent to “There exists a

unique individual x whose life span starts at time t such that…" However, since the statement refers to a specific new individual the system can create a constant name for the individual and reduce the expression to a proposition by substituting this name for the quantified variable, as it does for universal quantification. Thus, the `CREATED` quantifier is an existential quantifier that can be used where it would have a positive sense. Further, it generally does not make sense to use it where it would have a negative sense, and this is not permitted.

The second major difference between Gordius and predicate calculus is that Gordius relies heavily on *semantic attachment* [25] Semantic attachment (also known as procedural attachment) is the technique of using specially coded reasoning procedures to make inferences concerning certain predicates, rather than relying on axioms and the general rules of inference. A common example of semantic attachment concerns the treatment of numbers. One could perform all reasoning regarding natural numbers by using Peano's axioms of arithmetic. However, it is generally more useful and efficient to use the computer's built-in ability to represent and compute with numbers. Logic programming languages such as Prolog [26, 27] do precisely this.

Gordius uses semantic attachment for two purposes, both aimed at reducing the computational time-complexity of reasoning. The first purpose is to take advantage of the fact that inference with *iconic* (or analogic) representations is often much more efficient than formal manipulation of sentences in logic. An iconic representation is a representation that employs a data structure or procedure that has an isometric relationship to the logical relations of interest. For example, the ordinal relations '<,' '=' and '>' induce a lattice among quantities. Gordius employs a subsystem called the Quantity-lattice [28] to reason about such relationships among quantities. The quantity-lattice subsystem employs a graph data structure that explicitly represents the lattice. Inferences based on transitive closure over these relations can be computed efficiently by traversing this data structure.

Objects called layers and regions are used to represent the structures built up on a wafer during manufacturing. Semantic attachment is used to keep track of time-varying relations among these, such as what layer is immediately above another at a given time. It is more efficient to examine the data structure employed by this semantic attachment to determine whether a given layer is above another at a given time than to construct a proof (or disproof) of that fact, which would have to examine all

manufacturing events occurring before the specified time. The data structures constitute a symbolic ‘diagram’ of the spatial relationships among the layers and regions they represent.

The second purpose for semantic attachment concerns equality reasoning. Formal manipulation of logic sentences requires, as an integral operation, the matching (unification) of terms. It is important to know when two terms represent the same individual. This is not trivial because all individuals can be referred to in many ways. For example, “Elsie” and “John’s mother” might represent the same individual, but these two expressions are syntactically different (in predicate logic as well as in English). For these two terms to be unified the system would have to prove that they represent the same individual, and then use substitution of equals to make the two statements being unified syntactically identical.

The Gordius system is built on another system, called the Reasoning Utilities Package [29]. This subsystem is capable of performing this equality reasoning. However, as one might expect, equality reasoning is very expensive. It requires an extensive search for ways to prove that two terms might be equivalent every time it attempts to match the terms. Matching terms, though, is a very frequent “inner loop” computation in logical reasoning. Thus DAS has the equality reasoning feature within Gordius turned off. Instead, where equality reasoning is important DAS relies on semantic attachment. Every individual that DAS reasons about is represented by a Lisp object in the ‘diagram’ due to semantic attachment. This means that every individual in the universe of discourse has a canonical name. For those circumstances where equality reasoning is important, DAS uses semantic attachment to evaluate terms, determine the individuals they represent, and substitute their canonical names.

For functions representing time-varying attributes of objects, semantic attachment is automatic in Gordius, due to the object-oriented type system mentioned above. For other predicates and functions, Gordius has two features for creating semantic attachments. These are called *generic-predicates* and *generic-functions*. By declaring a predicate to be a generic-predicate, one can provide arbitrary Lisp code to implement that predicate. A prime use of generic-predicates is to perform the substitution of canonical names required as a substitute for equality reasoning.

While use of semantic attachment can improve efficiency, it also reduces confidence in the soundness of reasoning, as the reasoning procedure now incorporates arbitrary Lisp code, and its soundness depends on the validity of that code.

### 4.3.2. Elements of the Representational Machinery

#### 4.3.2.1. Temporal Objects and Attribute Functions

As mentioned in the previous subsection, the Gordius modeling language is a typed logic. Gordius provides several basic types, including various types of real numbers (for example `finite-positive-real`), time-points, time-intervals and sets. It also provides a mechanism for creating user-defined types. All user-defined types are subtypes directly or indirectly of the basic type '`temporal-object`.' Temporal-objects have a limited life span: each individual's existence starts at a specific time-point, and extends over a continuous interval of time, which may be indefinite in length.

The schema used to define new types of temporal objects also enables the user to specify a set of functions applicable to individuals of the type being defined. These functions specify time-varying attributes associated with these objects. There is a simple inheritance mechanism by which the attribute functions that are applicable to individuals of a given type also apply to individuals of any direct or indirect subtype. Intuitively, an individual to which an attribute function applies is considered to *have* that attribute.

Figure 4.1 below gives some examples of the schema for defining new types. The schema is implemented as a Lisp macro called defGDobject. The first argument of the macro is the name of the type being defined. The second argument is a list containing the name of the supertype of which this type is a subtype. Finally, the third argument is a list of attribute function specifications. Each attribute function specification is a list of the name of the attribute function and a specification of the type of the function's value. As shown in the figure, the type horizontal-region is a subtype of temporal-object to which the functions left, right, left-pos and right-pos apply. The values of the left and right attribute functions are individuals of type horizontal-region, and the values of the left-pos and right-pos attribute functions are finite real numbers. Thus, a horizontal-region object has attributes called left and right, which indicate what other horizontal-region object is to be found to the left and right, respectively, of the object at any given time. It also has numerical time-varying attributes indicating its left and right spatial extents.

```
;;; Class of objects that describe the lateral topology
;;; and dimensions for a single laterally homogeneous region
(defGDobject HORIZONTAL-REGION (temporal-object)
  ((left horizontal-region)    ;; neighbour
   (right horizontal-region)
   (left-pos finite-real)      ;; position
   (right-pos finite-real)))

;;; Class of objects that describe the lateral topology
;;; and dimensions for a single laterally homogeneous region
;;; of a 2D slice through a structure on a wafer
(defGDobject WAFER-REGION (horizontal-region)
  ((region-type region-type)
   (pieces (set . wafer-region))))

;;; Class of objects that describe the lateral topology
;;; and dimensions for a single laterally homogeneous region
;;; of a mask for a 2D slice through a structure
(defGDobject MASK-REGION (horizontal-region)
  ((left-wafer-region wafer-region)
   (right-wafer-region wafer-region)
   (opacity (one-of (constant opaque)
                    (constant transparent)))))
```

Figure 4.1 Example type definitions

The wafer-region type and the mask-region type are both indicated as subtypes of the horizontal-region type. Thus, the attribute functions that apply to horizontal-region individuals also apply to wafer-region and mask-region individuals. In addition, wafer-region objects have an attribute called region-type whose value is an object of type region-type, and an attribute called pieces whose value is a set of the sub-regions that the region might be subdivided into. The expression given as the value type for the opacity attribute of mask-region objects indicates that the value of this attribute must always be one of the two specified constant values 'opaque' and 'transparent.'

The value of any attribute function is actually a *history.* A history specifies the value of the attribute as a function of time. It is similar in concept to a *fluent* [23]. A history is composed of a sequence of time intervals, alternating between two types: *dynamic* and *quiescent*. A dynamic time interval is a time interval during which the value of the attribute undergoes some change. The bounds of a dynamic time interval are determined by the time interval during which the action responsible for the change occurs. The value of the attribute is considered to be known only at the time points that begin and end the interval. A quiescent interval is a time interval during which the value

of the attribute is not changing. Thus, the value of the attribute is the same at all time points within the interval.

The function '`@`' is used to extract the value of an object's attribute at a given time point. Thus, if '`mr-1`' is a mask-region, then the mask-region to its left at time '`t`' is '`(@ (left mr-1) t)`.' When the type of the value of an attribute is itself a subtype of temporal-object, then this syntax can be nested. The opacity of the mask-region to the left of `mr-1` at time `t` would be

'`(@ (opacity (@ (left mr-1) t)) t)`'

To simplify expressions, Gordius permits elision of the interior applications of the `@` function when all applications refer to the same time point. Thus, the previous expression can be rewritten as '`(@ (opacity (left mr-1)) t)`.' The time points most commonly referred to are those that begin and end dynamic time intervals. The two functions '`start-of`' and '`end-of`' applied to a time interval express the time points that respectively begin and end the time interval.

#### 4.3.2.2. Action Descriptions

Modeling activity in Gordius is principally concerned with describing the effects of actions. These effects involve the creation and destruction of objects, and changes in the values of objects' attributes. The modeling language was originally developed for reasoning in the domain of geologic processes, and so for historical reasons actions are called 'processes.' Gordius provides a Lisp macro called defProcess that implements a schema for describing these processes. Figure 4.2 indicates the syntax used. The description may contain up to five components, each labelled with one of the five labels: 'ako,' 'preconditions,' 'parameters,' 'effects' and 'constraints.' Any component can be left out (though a process description without the effects component would not be very useful). To avoid confusion with other uses of the word 'process' in this document, we will continue to use the term 'action' for what Gordius calls 'process.'

Preconditions are statements that must hold so that it is possible for the action to occur. Such statements are useful when attempting to synthesize plans. During plan synthesis the preconditions of an action to be included in a plan become sub-goals that must be satisfied in the context in which the instance of the action is to occur. As this

dissertation is not concerned with synthesis, our models will not include descriptions of preconditions.

```
(defprocess <process-name>
  ako <super-process>
  preconditions <list of precondition statements>
  parameters <list of (<parameter-name> <parameter-type>)>
  effects <list of effects statements>
  constraints <list of constraints> )
```

Figure 4.2 Schema for describing processes (actions)

Actions can be parameterized. In that case the names of the parameters are explicitly listed in a separate component of the description. Associated with each parameter name is a specification of the type of the acceptable values for the parameter.

The effects component of the description is a list of statements that indicate the impact of an occurrence of the action. This corresponds to but extends the add and delete lists of traditional discrete action representations ([30-32]. These extensions enable the inclusion of effects that:

1. are expressed in terms which are relative to the input state (for example, “the thickness of layer L1 decreases by 5”);
2. are conditioned on the input state (for example, “if the layer’s material is silicon, then the thickness decreases”);
3. are universally quantified, allowing for world states with an a priori undetermined number of individuals (for example, “for all layers, the thickness decreases”); and
4. introduce new individuals, or terminate the existence of individuals.

Changes to the attributes of objects are described by the “change” statement:

```
(change <type> <attribute> <change> <action> <interval>)
```

The form indicates that the <attribute> is affected by the <action> which occurred during the <interval>. The <action> and <interval> specifications can optionally be omitted, in which case the name of the instance of the action being described and the interval during which it occurs are assumed. <attribute> is an expression suitable as an argument to the ‘@’ function. It is an attribute function applied to an object. <type> is either ‘=,’ or an arithmetic operator (+, -, *, /). If it is ‘=,’ then

an absolute change is being specified, and after the action occurs the value of the <attribute> is equal to <change>. For example, the statement

```
(change = (above L1) THE-ENVIRONMENT ETCH1 I1)
```

represents the fact that after the ETCH1 instance of the ETCH action occurs during interval I1, the layer above L1 is the surrounding environment (that is, L1 is at the surface of the wafer, and no wafer layer is above it).

If <type> is an arithmetic operator, then a relative change is being specified and after the action occurs the value of the <attribute> is equal to the result of applying the operator to the initial value of the <attribute> and <change>. For example, the statement

```
(change - (thickness L1) 5 ETCH1 I1)
```

specifies that the ETCH1 event reduces the thickness of layer L1 by 5.

Change statements are only used where they would have a positive sense. One does not assert that a specific change doesn't happen, nor does one condition another assertion on the truth of a change statement.

Effects statements can be combined using the logical operators of predicate logic: conjunction, disjunction, negation, implication and equivalence. In Gordius, these operators are denoted in prefix form, following the Lisp idiom. Conjunction is represented by the logical operator '`:AND`,' disjunction by '`:OR`' and negation by '`:NOT`.' Conjunction and disjunction are n-ary rather than binary operators. Thus, if $S_1 \ldots S_n$ are acceptable effects statements, then so are `(:AND` $S_1 \ldots S_n$`)` and `(:OR` $S_1 \ldots S_n$`)`.

There are two distinct notations for implication. The syntax for normal implication is: `(:=>` <antecedent> <consequent>`)`. This is generally used when both <antecedent> and <consequent> are relatively simple statements. When this form is used, reasoning can occur in either the forward direction (from truth of antecedent deduce truth of consequent) or the reverse (contrapositive) direction (from falsehood of consequent deduce falsehood of antecedent).

Often it is the case that the contrapositive inference is not useful because falsehood of the consequent will never be established. Gordius provides an alternate

syntax for implications that are only intended to be used in the forward direction. The syntax is identical to that for normal implication, except that the prefix operator is '`:IF`' (suggesting the programming-language IF-statement). When this form of implication is used, Gordius does not attempt to evaluate the truth of the consequent until the truth of the implication is established.

The syntax for logical equivalence is `(:IFF` $S_a$ $S_b$`)`. This statement asserts that the truth of $S_a$ is the same as that of $S_b$.

Gordius provides three versions of universal quantification. The first version quantifies over all individuals of a given type that exist at a given time. The syntax is:

`(FOR-ALL-EXISTING` <var> `:` <type> <time> <statement> `)`

<var> is the name of the quantified variable, <type> is the type of individual that <var> can be bound to, <time> is the time point at which the individuals that <var> can be bound to must exist, and <statement> is an effects statement in which <var> appears as a free variable. The statement indicates that <statement> is true of all objects of type <type> that exist at time <time>.

The second version is more general. It can either quantify over all objects of a given type or all members of a set (regardless of when these individuals exist). The syntax is:

`(FOR-ALL` <var> <: or e> <type or set> <statement> `)`

<: or e> is either '`:`' indicating quantification over a type, or '`e`,' (suggesting element) indicating quantification over the members of a set. <type or set> is either the name of a type or a specification of a set, as appropriate to the type of quantification indicated by <: or e>. This is the only form of universal quantification that can be used where it would have a negative sense, and only when the quantification is over a finite closed set. When the system is aware that it knows all the individuals being quantified over, it can establish the truth of the quantified statement from the truth of the <statement> for each binding of <var>.

The final version of universal quantification is similar to `FOR-ALL`, but realizes an efficiency. The syntax is identical except that the prefix is `FOR-ALL-TRUE`. The efficiency is that the <statement> is only evaluated when the truth of the quantified statement is established.

Two versions of existential quantification are provided. The syntax for the normal existential quantifier of predicate logic is:

(EXISTS <var> <: or e> <type or set> <statement>)

where the elements of the syntax are the same as for the `FOR-ALL` and `FOR-ALL-TRUE` universal quantifiers. The quantified statement states <statement> is true when <var> is replaced by some object(s) of type <type or set> or member(s) of the set <type or set>.

The second version of existential quantification allows for the introduction of a single new individual. The syntax is:

(CREATED (<var> <type> <time>) <statements>)

This statement asserts that a new individual of type <type> begins to exist at time point <time>, and that the statements <statements> are true when <var> is replaced in them by the name of this new individual. Note that this quantifier quantifies any number of statements rather than a single statement. This syntax was chosen because of the frequency with which the quantified statement is a conjunction of many statements.

The termination of the life span of an object is indicated by the syntax:

(DESTROYED <object> <time>)

The constraints component of the process description contains statements that do not indicate changes produced by the action, but rather describe how the changes produced relate to one another. For example, a constraint might indicate that the total dopant in a layer is the same before and after a diffusion operation occurs. The

constraints component may also contain statements defining ‘macros’ that facilitate the writing of effects statements. The syntax:

(DEFN <name> (<var$_1$>…<var$_n$>) <form>)

is a constraint that introduces new syntax. It states that expressions of the form (<name> <val$_1$>…<val$_n$>) should be replaced by instances of <form>, with the values <val$_1$>…<val$_n$> substituted for the corresponding variables <var$_1$>…<var$_n$>. This feature can be used to create parameterized short-hand notation for frequently used expressions.

Finally, the ‘ako’ component of the description allows for a taxonomic organization and a simple form of inheritance among process descriptions. Ako is an acronym for “a kind of.” Its argument is the name of a more abstract process that the process being described is based upon. The inheritance is cumulative: the process being described inherits all preconditions, parameters, effects and constraints of the super-process and all its ancestors in the hierarchy, as well as those explicitly indicated in the corresponding components of the description.

#### 4.3.2.3. Generic Predicates and Functions

Along with the attribute functions defined by the specification of the object types that will be used to represent elements in the domain, one can introduce predicates and functions of arbitrary arity simply by giving them names and using them. Then the models of actions must contain statements relating to these functions and predicates that define what tuples of individuals satisfy the predicates and what values tuples of individuals are mapped to by the functions.

However, it is also possible to attach Lisp procedures to predicates and functions to help define them. Further, for the reasons mentioned in section 4.3.1.2 above, it is sometimes necessary to employ the semantic attachment features of Gordius to handle equality of terms through evaluation to canonical names for the individuals they represent.

Gordius provides two Lisp functions for performing this semantic attachment. The syntax for attaching a procedure to a predicate is:

```
(def-generic-predicate <predicate name>
  <list of (<argument name>  <argument type>)>
  <optional body>)
```

The first argument to the function is the name of the predicate. The second argument is a list specifying the arguments to the predicate. For each argument, a list of the argument's name and its type is specified. The remaining arguments form the optional body of Lisp code that defines the predicate.

The terms given as arguments to the predicate are evaluated to determine the Lisp objects representing the individuals that the terms represent, thus determining canonical names for the individuals. Furthermore, each tuple of arguments passed to the predicate are cached along with the truth-value of the predicate for that tuple, so that the truth value is immediately accessible for any equivalent tuple of terms. If the optional body is omitted, the truth-value is determined by the statements that employ the statement.

If the optional body is included, it should ultimately assert a truth value for the predicate for the tuple of arguments passed. Within this body of Lisp code, the name of the predicate is bound to the statement whose truth-value is being determined, and the symbols naming the arguments are bound to the Lisp objects representing the values of the arguments. The function '`assert-expr`' can be used to assert the truth-value determined. Truth-values are specified by the keyword symbols `:TRUE`, `:FALSE` and `:UNKNOWN`.

The syntax for attaching a procedure to a function is:

```
(def-generic-function <function name>  <arguments>  <value-type>
  <optional body>)
```

The first argument is the name of the function. The second is a list of argument specifications, as for the def-generic-predicate function. The third argument specifies the type of the value of the function. The remaining arguments form the optional body of Lisp code that defines the function.

The arguments to any call on the function are evaluated, and the tuple of Lisp objects is cached along with the corresponding value. As in the case of predicates, the Lisp code is optional. If omitted, the value of the term is determined by the statements that employ it.

If the optional body is included, it should use the assert-expr function to assert equality between the term being evaluated (to which the symbol naming the function is bound within the body) and the term identifying the value. If a precise value cannot be determined, it is also permissible for the body to make assertions that form a partial description of the value. For example, if the type of the value is a real number, the body may make assertions regarding how the value is ordinally related to other numbers, thus bounding the value.

## 4.4. Symbolic Qualitative Modeling of Semiconductor Manufacturing

### 4.4.1. Goals of the modeling process

Although the models of semiconductor operations are structured around describing the impact of each operation, their primary purpose is not to support the simulation task of predicting the outcome of performing the operation. Indeed, as we shall see in the chapter on process representation below (Chapter 5), the representation of the manufacturing process used to drive the simulation will often contain explicit statements describing the expected impact of performing the operation. The purpose of these models will be to explicate the structure of the causal relationships in the manufacturing process, to support the diagnostic task. Thus, in designing the models, accurate prediction is not the main concern. Instead, the criterion to be satisfied is the ability to generate a causal explanation why the operation achieves the impact that it does.

In the context of supporting the use of generic rules (introduced in Chapter 6) to perform diagnosis, the purpose of performing the simulation is even simpler. The rules are about how anomalies in specific kinds of processing affect specific kinds of features of the product being manufactured (in our case, specific kinds of structures on the wafer). The purpose of the simulation is merely to perform the complex bookkeeping chore of keeping track of which operations are causally involved in the creation and modification of which features of the product.

Accomplishing this goal requires models with a certain degree of simplicity. They should employ the kind of naive conceptualization of what's going on in the manufacturing process that admits of such causal explanations. The models described here aspire to capture the "commonsense" model of semiconductor manufacturing reflected in the two-dimensional depictions of vertical cross-sections through structures on the wafer (such as figure 4.3) that semiconductor technologists typically draw for one another in their daily communications.

It should be recalled that the major reason for going to all this trouble is to achieve robustness in knowledge representation. The hope is that the same set of models can be reused to reason about many different manufacturing processes. This implies that the models must constitute a set of building blocks that can be put together in a variety of ways to represent these different processes. The models must be designed to work together, in the same sense that LEGO™[2] bricks and parts are designed to work together (for example, the form factors for LEGO™ windows always conform to the sizes and shapes of holes that can be created with the bricks). Thus, it is important to pick a conceptualization of the product and the manufacturing process and make the models of all operations adhere to it. In deciding whether and how to represent phenomena, one must consider how the decision affects the modeling of all operations represented.

### 4.4.2. Modeling the wafer

The models of operations describe the impact of the operations on the product being manufactured. Further, the product's state at the beginning of an operation can influence the impact of performing the operation. Distinct operations in the manufacturing process interact because they affect the same product. The effects of subsequent operations mitigate the causal relationships between an operation and features of the product. The product serves as the sole medium of communication for this interaction. Thus, the description of the product carries the burden of this communication during simulation.

---

[2]LEGO is a registered trademark of InterLEGO AG.

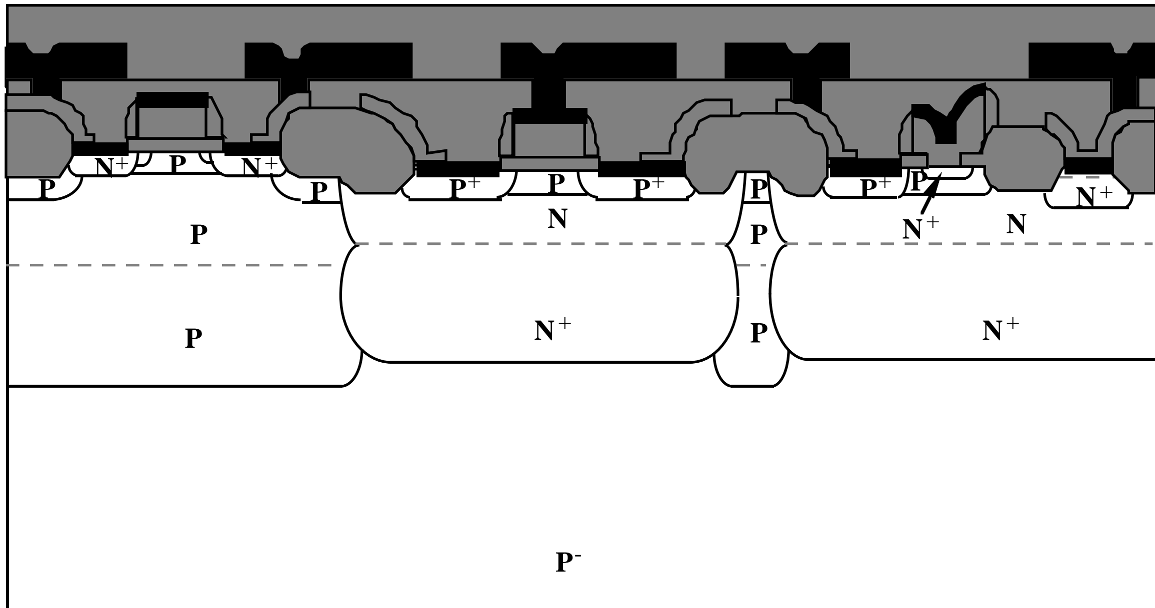


Figure 4.3 Sketch of Vertical Cross-Section Through BiCMOS Transistors
(from Prof. James Plummer, personal communication)

In the semiconductor domain, the product is the set of structures created on the wafers processed. In modern manufacturing processes, literally millions of similar structures are manufactured and connected into complex circuits. When reasoning about the behaviour of the product, all these structures must be represented in order to reason about the behaviour of the entire circuit. However, when reasoning about the manufacturing process, one only reasons about prototypical examples of the kinds of structures manufactured. One only recognizes that more than one instance of each structure is being created when one is investigating nonuniformity of processing distributed spatially across the wafer surface.

These structures are three-dimensional physical entities with potentially quite complex geometrical shapes. However, most are designed with two axes of symmetry, so that their geometries are adequately captured in two orthogonal "vertical" cross-sections through the wafer. Furthermore, most processing occurs "from above." The manufacturing operations have effects that are ideally vertical. The "lateral" effects of these operations are generally minimized.

For these reasons, the conceptual model adopted for describing wafer structures employs what might be called a 1½D approach. The wafer structures are represented by

a series of vertical strips. Each strip represents the sequence of layers one would encounter in drilling a hole through the center of a laterally homogeneous region of the structure. The strips retain their nominal lateral dimensions and the representation as a whole maintains the lateral topology of the structure: which types of region are side by side. However, no attempt is made to represent the complex geometry of the transitions between regions. This fact is emphasized in the graphical representations of the wafer-structure model by vertical gaps drawn between regions (Figure 4.4). This approach allows the structures on the wafer to be composed entirely of simple rectangles. One approach to creating a three-dimensional model of wafer structures using numerical simulation uses a similar technique: many one-dimensional simulations are performed simultaneously on a parallel computer and then juxtaposed [33].

This parallels the development of quantitative models of semiconductor phenomena. The mathematical models used by the earliest quantitative simulators only modeled a single, laterally homogeneous region of the wafer. Although two-dimensional mathematical models [34, 35], have been available now for several years, most of the quantitative simulations performed still use one-dimensional models. This is partly due to the computational complexity of these models, lack of faith in their accuracy, and the general difficulty of getting university research results into mainstream use. However, it is also the because the 1D simulations are sufficient for many purposes, and the value of the additional information provided by higher-dimension simulations does not warrant their additional computational expense.

Of course, such unavoidable lateral side effects cannot be ignored completely. They can have an important impact on device behaviour. Also, modern processes sometimes capitalize on these unavoidable effects by designing structures and processes that depend on them. Nonetheless, it is very difficult to reason symbolically about the inherently two-dimensional complex geometries that can occur at the transitions between regions. It is an open research problem to develop an ontology for symbolically describing two-dimensional shapes in a manner that facilitates reasoning about how these shapes affect and are affected by the physical and chemical processes involved in semiconductor manufacturing.

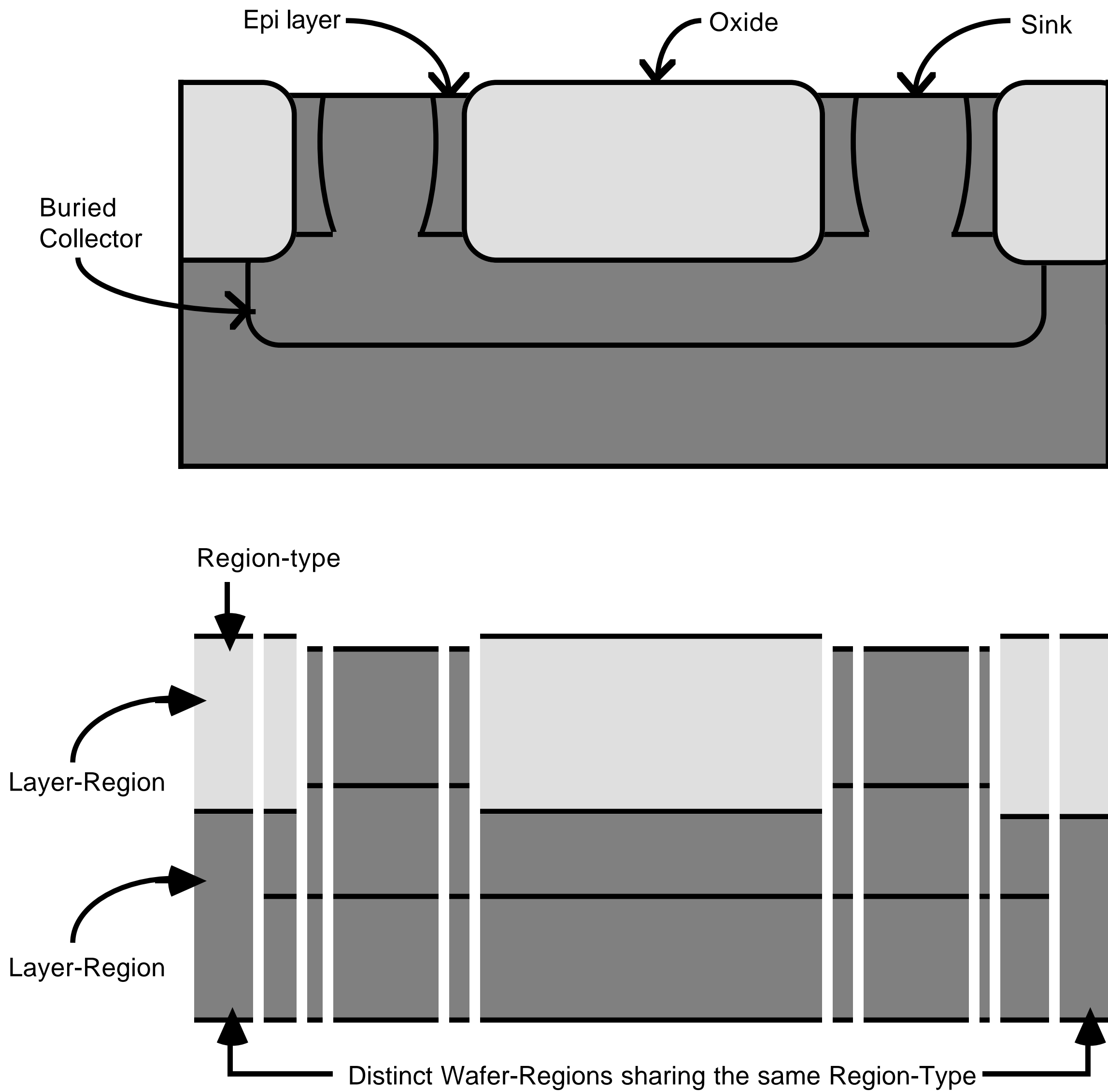


Figure 4.4 Cross-sectional drawing of a Collector-Under-Oxide resistor, and how DAS represents this same structure.

Semiconductor technologists conceptualize regions of doped silicon in two different ways. One way is to think of them as regions with well-defined boundaries with homogeneous concentration of a single dopant. The depictions they draw, such as Figure 4.3, reflect this conceptualization. Our models use this conceptualization. Doped regions form distinct layers just as regions of distinct materials do.

The other conceptualization recognizes that concentrations of dopant vary continuously, and that several dopants may exist simultaneously within the same region

of silicon. This conceptualization distinguishes between layers only when the basic material is distinct. All the contiguous crystalline silicon is considered to be a single layer, and dopant concentrations are represented by continuous curves called 'dopant profiles' that indicate the concentration of each dopant species as a function of depth below the surface of the layer. This alternate conceptualization is not considered in this dissertation.

The actual definitions of the object types used to describe wafer structure as well as processing materials is given in Appendix A. There is a hierarchical, part-whole flavour to the representation. At the leaves of the hierarchy, the fundamental unit of representation for geometry is called a **layer-region**. The name is chosen to reflect that the object represents a single layer in a specific type of region. These layer-region's are grouped into vertical columns. Each is represented by an object called a **region-type**. Each layer-region has attributes indicating which layer-region is **above** and **below** it, as well as numerical attributes for the positions of its vertical boundaries (**top-pos** and **bottom-pos**). For layer-regions at the wafer surfaces, the value of the above (below) attribute is a constant called **THE-ENVIRONMENT**.

The region-type objects represent the different types of regions created due to lateral differential processing. That is, processing operations that affect distinct lateral regions in different ways. These objects have only two attributes. One called **surface** indicates what layer-region lies at the wafer's top surface in regions of its type. The other, called **regions**, is the set of wafer-regions that wherein the region-type occurs. Employing the region-type concept realizes an efficiency: the vertical topology and geometry of layers in each type of region are represented only once, regardless of how many times such types of region occur in the structures being simulated.

The lateral topology and nominal lateral geometry of the structures being simulated is captured by objects of type **wafer-region**, which is a specialization of **horizontal-region**. Another specialization of horizontal-region, **mask-region**, is employed in the representation of the geometry of the masks used in photolithography operations. These object types were shown in Figure 4.1 above. A horizontal-region has attributes for the regions that neighbour it to the **left** and **right**, and for the positions of its lateral edges, **left-pos** and **right-pos**. A wafer-region also has an attribute for its region-type, and another for the sub-regions into which it might have been differentiated. All lateral differentiation in semiconductor processing can be traced

ultimately to photolithography operations that use masks to expose only certain parts of the wafer. Thus, in our simple model, all lateral edges of wafer-regions will be determined by the edges of mask-regions.

Cut-lines through individual structures to be simulated are represented by objects of type **2D-structure**. These have attributes for the **horizontal-regions** (a set) comprising them, and the **left-most** region in that set. Finally, there is an object type called **wafer** which has an attribute indicating what 2D-structure's appear on the wafer being simulated.

In addition to the attributes for describing geometry and topology, layer-region objects have attributes concerned with describing intrinsic properties. These include: **material**, **dopant** and **concentration**. The material attribute encodes the basic material of the layer, i.e., its chemical composition. The dopant attribute indicates the dominant species of impurity that the layer contains, if any. The concentration attribute indicates the concentration of the dominant impurity, over and above the concentration of impurities of opposite polarity.

Finally, layer-regions have two additional attributes. The **region-type** attribute specifies the region-type to which the layer-region belongs. The **layer** attribute specifies the layer to which the layer-region belongs.

What is a layer? The notion of layer is prevalent in the discussions of semiconductor technologists, but it is not anywhere well-defined. The terminology stems from the imagery of integrated circuit structures being built up in layers like those of a layer-cake. It is reinforced by the fact that materials are often deposited on the surface of the wafer in layers that blanket the wafer uniformly (though they are generally subsequently patterned by photolithography operations).

Consider the case of the 'field oxide.' This is a very thick layer of silicon dioxide that is grown between areas where active devices will be situated as a form of insulation between the devices. All pieces of the field oxide are created at the same time, and have the same physical properties. However, the field oxide is restricted to growing only in the areas between active device sites by covering the active device sites with silicon nitride. This material oxidizes much more slowly than does crystalline silicon, so that no thick oxide forms in those areas covered by it. Nonetheless, silicon nitride does oxidize.

This implies that an oxide grows on top of the nitride as well as on top of the silicon in exposed areas. Further, this oxide is contiguous with the field oxide, and is made of the same material: silicon dioxide. Yet, the oxide grown on the nitride and the oxide grown on the exposed silicon are not considered to be part of the same layer. The oxide on the nitride is 'inconsequential,' and will be stripped away along with the nitride, whereas the field oxide will remain as a part of the product.

The effects of processing operations are most easily described in terms of their impact on the individual layer-regions of the DAS wafer representation. However, it will be beneficial, especially when supporting the development of generic rules, to capture some version of the layer notion. The DAS representation represents layers by collecting layer-regions into sets. Layer-regions must have the same intrinsic physical properties and be created at the same time to be considered part of the same layer. Further, they must be created in a 'similar context,' where the meaning of this varies from one operation to the next. In oxidation steps, for example, the resulting oxide layer-regions will be part of the same layer only if they were created by oxidizing layer-regions that were initially part of the same layer. This definition comes close to the informal notion of layer in engineers' parlance, but will differ from it primarily by making additional distinctions. For example, not all 'field-oxide' layer-regions will be collected together into a single layer, if some of them are created by oxidizing silicon in a p-doped region and others are created by oxidizing silicon in an n-doped region. This is because the differently doped regions will necessarily not be part of the same layer.

DAS employs the object type **layer** to represent layers. Objects of this type have one attribute, **layer-regions**, that is the set of layer-regions forming the layer. Since layers are sets of layer-regions, and the intrinsic physical properties shared among these layer-regions are attributes of the layer-regions themselves, an artifact of the representation is that one has to select a specific layer-region to determine these physical properties. To overcome this difficulty, DAS provides the function **any-member**, which maps from a set to an arbitrarily selected member of the set.

In the course of simulating a process, the DAS simulator will introduce many new objects representing layer-regions, layers, region-types and regions, as they are discovered to be created by the effects of processing. These objects will be assigned unique names of the form <type>-<number> (e.g., layer-region-16). It is often necessary to refer to parts of the structure being manufactured in the description of the

manufacturing process. The names that will be assigned to these objects cannot be predicted in advance. Multiple simulations will assign different names. Thus, one has to refer to these objects by description rather than by name. For example, “the layer-regions at the surface in the regions that were not covered by photoresist after step X.” This practice can be quite cumbersome. Engineers often assign names to layers that reflect the function that the layer performs either in the product, or in the process of creating the product. To facilitate reference to parts of the structure being manufactured, DAS provides a predicate, **named**, for assigning user-defined names to objects. Also, DAS provides a predicate, **created-during**, which facilitates describing objects by the time of their creation.

### 4.4.3. The basic kinds of steps modeled

The manufacturing plans used in the semiconductor industry can contain several hundred steps (300 is a typical for a modern process). However, all these steps are drawn from a relatively small repertoire of classes of operations. Our models are sufficiently abstract that we can capture a large subset of all these operations in only sixteen classes (four additional classes abstract the commonality among some of the sixteen). Figure 4.5 shows the operations we chose to model. Those in shaded rectangles are incomplete models: they are not executable. Lines in this figure indicate models related by the AKO relationship. The Photo-Resist-Develop and Photo-Resist-Clean step might have been modeled as specializations of the Etch step, but were not because the specializations were much simpler than the subsuming abstract model. The hierarchy in the diagram does not reflect the organization in the list below, because its organizing principle reflects the utility of inheriting model descriptions.

Most operations in a semiconductor manufacturing plan will be instances or specializations of the twelve listed below. The operations can be grouped into four categories as follows:

**Addition of Material:** these operations cover the upper surface of the wafer with a “blanket” layer of some material, although factors governing adhesion may allow for selectively depositing material only in certain areas. The Add-Top-Layer model abstracts the commonalities among these operations.

**Spin-on-Resist** — coats the wafer with a positive or negative photoresist.

**Sputtering** — coats the wafer with a layer of metal (e.g. aluminum).

**Chemical-Vapor-Deposition** — deposits silicon compounds such as silicon dioxide and silicon nitride, as well as other materials, such as tungsten.

**Epitaxial-Growth** — grows a layer of crystalline silicon, possibly doped with an impurity, as an extension of the crystalline silicon at the wafer surface.

**Removal of Material:** these operations selectively remove material from the upper surface of the wafer, depending on the material's characteristics.

**Etch** — removes those materials at the surface of the wafer that react with the particular etchant used.

**Photo-Resist-Clean** — removes all photoresist on the wafer.

**Photo-Resist-Develop** — removes only "soft" photoresist (positive photoresist that has been illuminated, or negative photoresist that has not been illuminated).

**Change of Chemical Properties:** these modify certain chemical properties of layers existing at the surface of the wafer.

**Oxidation** — combines silicon and/or silicon compounds with oxygen to form silicon dioxide.

**Mask-Expose** — changes the "hardness" of photoresist by using light or other radiation to alter chemical bonds. The radiation is patterned with a mask that indicates how the wafer surface is to be differentiated into distinct lateral regions.

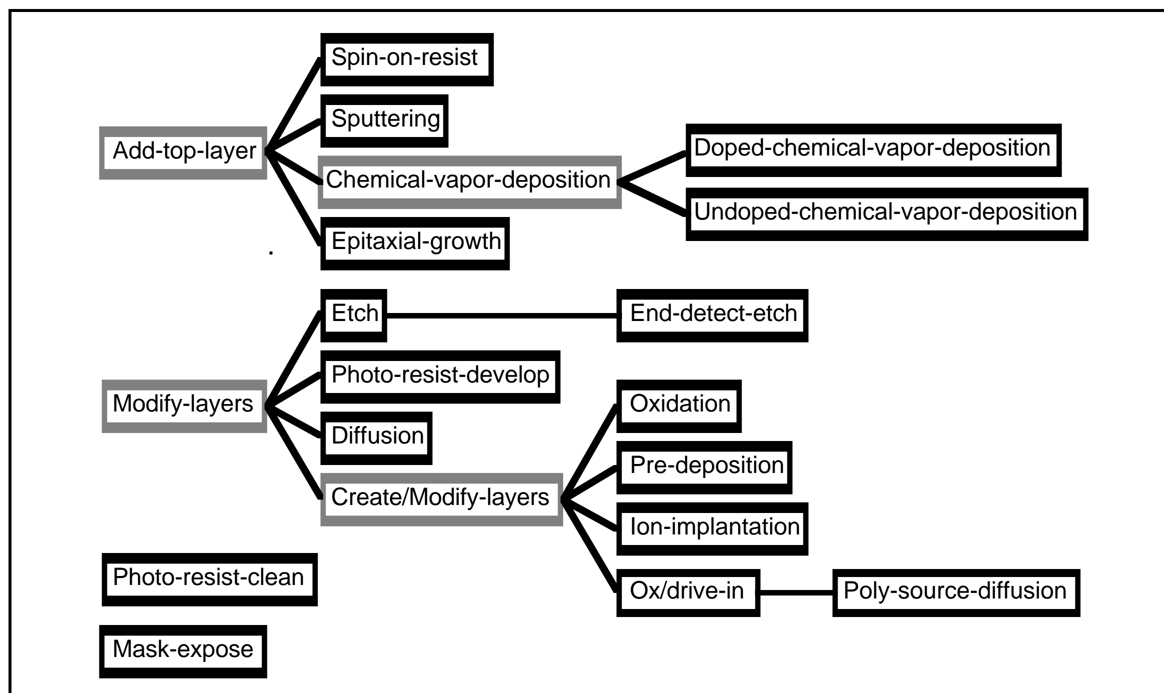


Figure 4.5 The basic operations modeled in DAS

**Change in Doping Profile:** these operations introduce impurities into the silicon crystal lattice and/or modify their distribution. Control over the distribution of impurities is the key to the formation of electronic devices.

**Ion-Implantation** — implants ions of a dopant below the surface of the wafer by accelerating them electromagnetically towards the wafer.

**Pre-Deposition** — introduces a very high concentration of impurity ions at the surface of the wafer. The resulting concentration is a function of the dopant type and the material of the layer, called the solid solubility of the dopant in that material.

**Diffusion** — modifies the distribution of impurity ions by heating the wafer, enabling ions to diffuse.

There are several types of steps which are not included in this set. We explicitly chose not to model operations that do not have consequences for the topology and geometry of structures on the wafer, such as cleaning steps and photoresist baking steps. Also, operations that act on the backside of the wafer, such as those performed to achieve a gettering effect, are not included because our models do not indicate the effects of any operations on the wafer's backside. (Gettering involves causing damage to the structure of the crystal lattice on the backside of the wafer, because such damage traps unwanted impurities, such as gold, thus reducing the concentration of such impurities in the rest of wafer.)

Each operation is modeled in the Gordius system as a discrete action (a *process* in Gordius terminology). The precise definitions of these models are recorded in Appendix B. Descriptions of the structure of these models appear in [7]. The one major facet of these models that is not described in that paper is the treatment of the layer concept. In the next two subsections we will discuss in detail the structure of the two models that have changed the most since the publication of [7]: the Etch and Diffusion models.

#### 4.4.3.1. Writing Inheritable Models: The Etch Models

The Etch operation is the conceptually simplest operation that nonetheless demonstrates many of the difficulties that must be overcome, and the techniques employed to overcome these difficulties. As the name suggests, etching in the semiconductor world is similar to etching and engraving in the world of arts and crafts. A chemical reaction is induced that causes material at the surface of the wafer to combine with a reagent to form a by-product that does not adhere to the wafer's surface. In this fashion, material at the surface of the wafer is incrementally removed.

The amount of material removed can depend on a variety factors. The most important are: the chemical composition of the material being removed (and its physical state, such as whether it is amorphous, crystalline or polycrystalline); the reagent used and its concentration; and the length of time the reaction is permitted to proceed. Since different materials react with the reagent at different rates, the speed at which material is

removed — the etch rate — is different for different layers. This makes it possible to create patterns in a layer through selective etching, and to transfer patterns in one layer into layers below it.

There are two basic subclasses of etching operations. In wet etch operations the etch reagent is dissolved in water, forming an acid in liquid form that the wafer is dipped into. Dry etching techniques employ an etchant in the form of a plasma. Some dry etching techniques impose an electric field across the plasma that causes ions in the plasma to drift toward the wafer surface. In dry etching operations, the etch rate also depends on the characteristics of this plasma and the strength of any such applied electric field.

In all these different types of etching techniques the activity taking place is fundamentally the same. Material is incrementally being removed from the surface of the wafer. Thus, we want to capture this commonality in our most abstract model in a manner that allows it to be shared among all etch techniques. To do this, we assume a simple linear model for all etch techniques: that is, we assume that the etch rate is constant throughout the operation for any given material (though it will, in general, vary from material to material).

If the wafer structure before the operation was a single homogeneous block, the impact of any etch step would simply be to reduce the thickness of the block. The amount by which the thickness was reduced would be the product of the etch-rate and the duration of the step. However, things aren't quite that simple. Our model must be general enough to handle any wafer structure presented to the step. Wafer structures in our representation can have any number of wafer-regions, any number of region-types (subject to the constraint that the number of region-types is less than or equal to the number of wafer-regions) and any number of layer-regions within each region-type. Further, the layer-regions can have any thickness, any chemical composition known to the system and any values for any other characteristics we ascribe to layer-regions. Finally, we must contend with the possibility that the etch step might etch completely through one or more layers in any given region-type, thus changing the topology of the wafer structure.

Since the number of components to the wafer structure is unknown, our model must use universal quantification to state how each component is affected by the

operation. As we do not model lateral effects, the etch step cannot have any impact on the topology or geometry of wafer-regions. The fact that etching starts at the surface of the wafer and proceeds downward suggests that we should quantify over region-types (to handle the fact that the effect of the step is different for each region-type) and somehow sequentially process the layer-regions in each region-type starting with the one at the surface and progressing downward to each layer-region ultimately affected by the step.

Unfortunately, logic does not readily lend itself to such sequential processing. We can accomplish the effect of sequentially processing the layer-regions in a region-type by keeping track of an incrementally growing set of the layer-regions affected by the step, and quantifying over the members of that set. The set of affected layer-regions would initially contain those layer-regions at the surface of each region-type. Each time it is determined that a layer-region is etched completely through, the layer-region below it is declared to be also a member of the set.

However, it is simpler to quantify over all layer-regions. The trade-off is that this approach instantiates the effects statements for all layer-regions whether they are actually affected by the step or not, but it avoids the necessity to create and manage an explicit set of those layer-regions affected by the step.

For each layer-region *l* the model makes two conditional statements:

1. the layer-region *l* is destroyed if and only if the duration of the step is longer than the time it takes to etch all the way from the surface of the wafer completely through the layer-region;
2. if the duration of the step is greater than or equal to the time it takes to completely etch through the layer-region directly above *l*, and less than the time it takes to completely etch through *l*, then
   a) if layer-region *l* wasn't at the surface before the step then the layer-region at the surface of the region-type containing *l* has changed to *l*, and THE-ENVIRONMENT has become the layer-region above *l*; also
   b) if the duration of the step is strictly greater than the time it takes to completely etch through the layer-region above *l* and this etch step etches layer-region *l* at a rate greater than zero then both the top-position and the thickness of layer-region *l* are reduced.

These two statements cover all the possible ways that a layer-region can be affected by the etch step, and explicitly indicate the changes in topology (does the layer-

region continue to exist? what is above it? what layer-region is at the surface?) and geometry (how thick is the layer-region? where is its top-position?) that might be involved.

The expression “the time it takes to etch all the way from the surface of the wafer completely through the layer-region” is represented in the model by a function called **Etch-Destroy-Time**. This function has a recursive definition. It is the sum of the quotient of the layer-region’s thickness and the etch-rate, and the etch-destroy-time for the layer-region above. The recursion bottoms out when the layer-region is THE-ENVIRONMENT. This recursive definition is provided by making the function a generic-function, with a semantic attachment that declares the function’s value to be equal to the value of the expression described above.

Note that the discrete action nature of the model requires that we state the ultimate impact directly in terms of the initial state and the step’s parameters. This makes it awkward to model events that have an obvious internal time structure. In our etch model, for example, the times when the etch process completely etches through a layer-region are natural landmarks. It would be more natural to give these times names and refer to them, or at least to give names to the sub-intervals indicating the time remaining of the duration as each layer-region is destroyed in turn. Then, etch-destroy-time could be a non-recursive function which refers just to the time it takes to etch through a given layer-region. While it is possible to do this, it is somewhat clumsy (it requires additional statements, for example). Also, the alternate approach we took has some advantages.

In particular, the recursive etch-destroy-time function facilitates writing statements that indicate what the process designer’s intention for the step is, and that constrain the expected outcome of the step. Such constraints are necessary when one simulates the step without quantitative information about the thickness of layers, the duration of the step, or the expected etch rates. It is possible to give quantitative values to the step parameters and wafer-structure attributes and have the system perform arithmetic. However, the design of the system also enables it to reason qualitatively. Reasoning qualitatively makes it possible to simulate a manufacturing plan before quantitative values have been assigned for all parameters and to obtain qualitative answers to incompletely specified questions.

```
(defprocess ETCH
 AKO Modify-Layers
 parameters
 (($etchant etchant-type) ($duration positive-real))
 effects
 ((for-all-existing $l : layer-region (SPI)
   (:AND
    (:IFF (>= $duration (etch-destroy-time $l (PI)))
          (destroyed $l (EPI)))
    (:=>
     (:AND
      (>= $duration
          (etch-destroy-time (@ (above $l) (SPI))) (PI)))
      (< $duration (etch-destroy-time $l (PI))))
     (:AND
      (:=>
       (:NOT
        (V= $l (@ (surface (region-type $l)) (SPI)))))
       (:AND
        (change = (surface (@ (region-type $l) (SPI))))
                $l)
        (change = (above $l) THE-ENVIRONMENT)))
      (:=>
       (:AND
         (> $duration (etch-destroy-time
                         (@ (above $l) (SPI))) (PI)))
         (> (etch-rate $l (PI)) 0.0))
       (:AND
        (change - (top-pos $l) (amount-etched $l))
        (change - (thickness $l) (amount-etched $l))))))
    (:=>
     (destroyed $l (EPI)))
     (member (@ (layer $l) (SPI)))
             (changed-layers (PI))))))
 constraints
 ((DEFN PI () (process interval))
  (DEFN SPI () (start-of (process interval)))
  (DEFN EPI () (end-of (process interval)))
  (DEFN AMOUNT-ETCHED ($l)
   (* (- $duration
         (etch-destroy-time (@ (above $l) (SPI))) (PI)))
      (etch-rate $l (PI))))))
```

Figure 4.6 The Basic Etch Model

The description of the etch model given above is not quite complete, as it does not specify the impact of the step on layers (as opposed to layer-regions). A side-effect of destroying a layer-region is that the layer containing it is changed. If all layer-regions in a layer are destroyed, then the layer itself is destroyed. To handle the impact on layers, one additional statement is included in the statement quantified over layer-regions:

3. if layer-region *l* is destroyed then the layer containing layer-region *l* is a member of the set of layers changed by the step.

Figure 4.6 shows how Etch Model statements are actually encoded in DAS.

A separate statement quantifies over the set of layers changed by the step. This statement is inherited from the Modify-Layers model. It has three conjuncts that summarize all ways that the layer might have changed. For every layer changed by the step:

1. for every layer-region in the layer at the beginning of the step, the layer-region is in the set called the new-layer-regions of the layer if and only if the layer-region survives the step;
2. if all the layer-regions in the layer are destroyed by the end of the step, then the layer itself is destroyed; and
3. if the layer is not destroyed, then the layer-regions in the layer at the end of the step are the layer-regions in the set called new-layer-regions of the layer.

The first conjunct collects all the surviving layer-regions into a set. The second conjunct destroys the layer if this set is empty, indicating that all layer-regions in the layer were destroyed. The third conjunct updates the history of the layer-regions in the layer if the layer survives the step. The use of the intermediary set allows us to avoid referring to the set of layer-regions comprising a layer at a time when the layer might not exist.

Figure 4.7 shows the encoding of the Modify-Layers model.

```
(defprocess MODIFY-LAYERS
 effects
 ((for-all-true $l e (changed-layers (PI))
    (:AND
     (for-all-true $lr e (@ (layer-regions $l) (SPI))
       (:IFF (:NOT (destroyed $lr (EPI)))
             (member $lr (new-layer-regions $l (PI)))))
     (:IFF (for-all $lr e (@ (layer-regions $l) (SPI))
             (destroyed $lr (EPI)))
           (destroyed $l (EPI)))
     (:=> (:NOT (destroyed $l (EPI)))
          (change = (layer-regions $l)
                  (new-layer-regions $l (PI)))))))
 constraints
 ((DEFN PI () (process interval))
  (DEFN SPI () (start-of (process interval)))
  (DEFN EPI () (end-of (process interval)))))
```

Figure 4.7 Abstract model inherited by models in which layers can be changed

In [7] , etch-rate was considered to be a function of the etchant used and the material of the layer being etched. This approach did not allow for the possibility of indicating the dependence of etch rates on other factors, such as dopant species and concentration, and other parameters of the etch step, such as those that determine or describe characteristics of the plasma in a dry etch step. Further, the factors influencing etch rate are different for different etching techniques, so that it is not reasonable to expect that we can specify a set of arguments for a general etch-rate function that will satisfy all cases.

In our current etch model, rather than attempting to list all possible influences on etch rate as arguments of a comprehensive Etch-Rate function, Etch-Rate is a function only of the Layer-Region being etched, and the time-interval of the etch step. These two arguments provide “handles” for getting at other factors determining etch rate. The intention is that the functional dependence of the etch rate on these other factors be specified in rules associated with the specific etching technique being modeled (perhaps even with the specific instance of the etching technique within a process).

In the event that use of a given etchant necessarily implies a unique etching technique and therefore a fixed set of factors influencing rate for all etch steps using that etchant, a general rule can be written independently of specific etch steps. A macro, DefEtchRateRule facilitates writing rules of this form. Within invocations of this macro, the statement is implicitly quantified over all layer-regions and all etch step time-intervals. Figure 4.8 shows an example of how qualitative information regarding the etchant “Nitric Acid” might be specified.

```
;;; Nitric Acid etches silicon, nitride and oxide, but not photoresist
(defEtchRateRule Nitric-Acid ($L $I)
  (:AND
   (:=> (:OR (is-type silicon
                      (@ (material $L) (start-of $I)))
             (is-type nitride
                      (@ (material $L) (start-of $I)))
             (is-type oxide
                      (@ (material $L) (start-of $I))))
        (> (etch-rate $L $I) 0.0))
   (:=> (is-type photo-resist
                 (@ (material $L) (start-of $I)))
        (N= (etch-rate $L $I) 0.0))))
```

Figure 4.8 Qualitative example of a general Etch-Rate rule

More generally, however, specific rules which apply only to individual etch steps will be attached to those etch steps. By this approach, all etch steps can inherit the same description of their behaviour, written in terms of the Etch-Rate function, but the nature of the dependence of etch rate on other factors can be individually specified for each type of etch step. The information provided may be as specific as the actual etch rate expected for the materials that the step is intended to etch, the “selectivity” (the ratio of the etch-rates for the material to be etched and the material below it or the material masking the etch), or it may specify an expression or function that gives the etch-rate for a wide class of materials.

One special subclass of etching techniques might be called “End-Stopped Etching.” End-stopped etch steps take advantage of the fact that some materials are almost impervious to some etch techniques. When one wants to completely remove a layer at the surface of the wafer, one chooses an etch technique that easily etches the material of the layer to be removed, but has little impact on the material of the layer beneath it. When it is possible to do this, one realizes the advantages that the etch step is less sensitive to the duration of the step, and that one can remove a layer that has different thicknesses in different regions. In our basic etch model, although the etch-rate is considered to be constant throughout any given layer, it can be different for each layer. Thus the basic etch model handles end-stopped etching without modification.

In a related but distinct etching technique, which we call “End-Detect Etching,” a feedback mechanism is employed that enables the etch process to proceed until a specific type of layer is exposed. While the duration parameter of the basic Etch model is considered an independent variable that determines how much etching takes place, in an End-Detect-Etch step the duration parameter is a dependent variable, determined by the structure of the wafer. Usually, an additional parameter, which we call Overetch, indicates how long the etch process should continue beyond the exposure of the detected layer, as a function of the time it took to expose the layer. The feedback mechanism reduces sensitivity of the process to actual etch-rates.

Our End-Detect-Etch model inherits from the Etch model. Modeling this sort of operation poses a technical difficulty. The duration of the step is (apart from the overetch stage) is determined by the length of time it takes to etch through the layers above the layer that is detected. This time will in general be different in different regions

— for example, it would be extremely long in regions covered by a layer that masked the etch. Thus, the model must select the regions for which this time is minimal.

This requires a universal quantification over all region-types in which the detected layer is present, that states that the time to etch down to the detected layer in a specific region-type is less than or equal to the corresponding time for all other region-types.

Unfortunately, the truth of such a statement can never be established because the set being quantified over is an open set. Its members are discovered incrementally as a consequence of processing operations, particularly photolithography operations. Thus, there is no point during the reasoning computation at which the modeling system knows that all such region-types have been identified. In fact, we present the manufacturing operations to the system in chronological order, so that the region-types it knows about at the time of the etch step are the only ones to be concerned with. The modeling system, however, does not “know” this. Note, furthermore, that this information lies outside the domain being modeled. It concerns the simulation computation, not the manufacturing plan being simulated.

To resolve this problem, we introduce a new generic function: Layer-Closure. The Layer-Closure of a layer is the closed set of layer-regions known to be members of the layer at the time the value of the expression is needed. Thus, by semantic attachment, the layer-closure function takes the extra-logical step of making a closed-world assumption, justified by the chronological presentation of the manufacturing operations.

The layer-regions for which the etch-until-exposed time is minimal are collected into a set called the Detected-Layer-Regions of the step interval. Another generic function, Any-Member, selects an arbitrary member of this set as the determiner of the etch duration. By reference to this set, the process specifier can provide constraints helping to identify the appropriate layer-regions in the absence of quantitative information about etch rates and layer thicknesses.

Figure 4.9 shows the encoding of the End-Detect-Etch model. It has three effects statements. The first specifies that none of the layer-regions in the detected-layer are destroyed, and that only those for which the etch-destroy-time of the layer-region above

is minimal are members of the detected-layer-regions set. The second statement indicates that the vertical-etch-duration is the time it takes to etch down to any member of the detected-layer-regions set. Finally, the last statement indicates that the total duration of the step is the vertical-etch-duration plus the overetch time. Of course, this model also inherits the statements of the Etch model and the Modify-Layers model.

```
(defprocess END-DETECT-ETCH
 AKO Etch
 parameters
 (($overetch positive-real) ($detected-layer layer))
 effects
 ((for-all-true $lr1 e (@ (layer-regions $detected-layer)
                          (SPI))
   (:AND
    ;; Detected layer is not etched through.
    (< $duration (etch-destroy-time $lr1 (PI)))
    ;; Layer regions "closest" to surface (in terms of etch time) are detected.
    (:IFF
     (for-all $lr2 e (layer-closure $detected-layer (SPI))
      (<= (etch-destroy-time (@ (above $lr1)(SPI)) (PI))
          (etch-destroy-time (@ (above $lr2)(SPI)) (PI))))
     (member $lr1 (detected-layer-regions (PI))))))
  (N= (vertical-etch-duration (PI))
      (etch-destroy-time
       (@ (above
           (any-member (detected-layer-regions (PI))))
          (SPI))
       (PI)))
  (N= $duration
      (+ (vertical-etch-duration (PI))
         (* $overetch (vertical-etch-duration (PI))))))
 constraints
 ((DEFN PI () (process interval))
  (DEFN SPI () (start-of (process interval)))))
```

Figure 4.9 Model of End-Detect-Etch

#### 4.4.3.2. Sacrificing Completeness to Limit Complexity: The Diffusion Models

Section 4.4.2 above points out that we use a very simplified representation of the distribution of impurities within the wafer: each layer-region is considered to contain a constant concentration of only one dopant. This representation matches the way wafer structures are often depicted in cross-sectional diagrams. Additionally, the representation (as presented in [7]) initially restricted the presence of dopants to layer-regions of crystalline silicon. This captured the intuition that, for the most part, the presence of dopants only has interesting consequences in crystalline silicon, and was sufficient for capturing the early bipolar process we were representing at the time.

The assumption that dopants in other types of layers can be ignored is not valid for modern MOS, CMOS and BiCMOS processes, in particular for the Stanford BiCMOS process, for two reasons. First, polysilicon (polycrystalline silicon) is often used as "wire" and as the gate material in the MOS transistors. This material is often doped to reduce its resistivity. It is therefore important to capture the impact of implantation steps aimed at controlling the resistivity of this material.

Second, the Stanford BiCMOS process employs a somewhat subtle technique to minimize the number of steps in the process. A single ion implantation step introduces dopant both into polysilicon regions that will be gates for NMOS transistors, and into polysilicon regions sitting over the active area of the NPN bipolar transistors. In the gate regions, the impact is to reduce the resistivity of the gate layer. The dopant in this layer is prevented from diffusing into the crystalline silicon substrate by the thin gate oxide layer interposed between the gate and the substrate. The dopant specie, arsenic, segregates preferentially into polysilicon over oxide, and so the oxide acts as a barrier. In the bipolar transistor regions, however, the polysilicon sits immediately above the crystalline silicon substrate, over the p-type base layer-region. During a diffusion step following the implantation step, the implanted ions diffuse across the boundary from the polysilicon into the silicon. The number of ions that diffuse across the boundary is sufficiently large that they outnumber the p-type ions in the base, and form a new n-type layer-region that becomes the emitter of the NPN transistor. Effectively, the doped polysilicon layer acts as a source of dopant for a pre-deposition step that creates the emitter.

To adequately capture the logic of the BiCMOS manufacturing process, the models of the wafer structure and of the ion-implantation and diffusion must allow for the presence of dopant in layers of all types, and for the diffusion of dopants across boundaries between dissimilar materials as well as within crystalline silicon.

Modifying the representation to allow for the presence of dopant in layer-regions of any material is trivial. The only complication stems from the fact that the presence of dopants can differentiate what was previously considered a single layer-region into several layer-regions. Semiconductor engineers generally do not differentiate layers according to doping except in the case of crystalline silicon. Thus, the representation deviates more from the intuition of the engineers.

Allowing for diffusion of impurities across boundaries between dissimilar materials is considerably more complicated. This is due to the discrete action nature of the models and to our simplified ontology for dopant profiles. The crux of the problem is the combinatorics involved in expressing all the possible scenarios and their outcomes. For each layer-region, one must consider whether the adjacent layer-regions are of the same or different materials (4 possibilities); whether they contain the same dopant species, another species or no dopant at all (25 possibilities, stemming from four commonly used dopant species and the no dopant option, squared due to two neighbouring layer-regions); and whether the concentration of dopants in the adjacent regions are higher or lower than the concentration of dopant within the layer-region (and for a dopant of opposite polarity, if higher then whether it is sufficiently higher for the influx of dopant to change the dominant species in the layer-region)(as many as another 25 possibilities).

These choices multiply, leading to 2500 possibilities that define most of the potential initial scenarios. Then one must consider all the possibilities for how long the diffusion operation continues. For example, in the case of a layer-region sandwiched between two layer-regions of different materials, when dopant is diffusing from one neighbour into the layer-region, is the layer-region sufficiently thin and the operation sufficiently long that the dopants also cross the second material boundary (i.e. is diffusion through a thin film taking place)?

A completely general model would have to consider all these possibilities because the discrete action, one-uniform-dopant representation does not permit the influence of each of these factors to be computed independently and then combined. There is no way to factor the problem into completely independent influences. However, some of this combinatorial explosion could be avoided by other representational choices. For example:

1. keeping track of the concentration of each dopant species independently rather than figuring out which specie dominates and representing only one per layer, avoids reasoning separately about each possible combination of dopants and their relative concentrations;
2. keeping track of dopant profiles rather than having dopant variations induce layer-region distinctions, avoids the necessity to identify situations that induce changes in wafer topology;

3. using an incremental evolution modeling approach (such as continuous process models) rather than a discrete action approach allows the many distinct outcomes of a situation to be represented implicitly rather than explicitly.

The representational choices in the above list are effectively how quantitative simulators avoid combinatoric complexity. While it is also possible to adopt these in a symbolic simulator, it involves developing machinery that is beyond the scope of this dissertation.

A very general model of diffusion is thus out of reach given our representational machinery and our ontological choices. However, we can still capture the causation in a modern process like the Stanford BiCMOS process by sacrificing generality. We accomplish this by writing specific models that capture only those phenomena that we expect to occur within a given step. For example, the diffusion step described above, in which the polysilicon layer above the NPN base layer acts as a dopant source for the creation of the emitter layer would be represented by a special Poly-Source-Diffusion model that incorporated the within-silicon diffusion effects of the original diffusion model by inheritance, and added effects statements that described the diffusion of the arsenic across the polysilicon/silicon boundary to form a new layer-region.

Figure 4.10 exhibits the Poly-Source-Diffusion model. The model's effects apply only when the material of a layer is polysilicon, the material of the layer below it is not polysilicon, and the concentration of the dopant in the first layer, multiplied by the segregation coefficient for the layer's dopant and the two layers' respective materials is greater than the concentration of dopant in the second layer. In that case a new layer is created representing the dopant that diffuses across the material boundary. The large number of "change" statements indicate the characteristics of this new layer, and the topological relationships among the layers. The model depends on three functions that define, respectively, the segregation coefficient for a given dopant and two materials; the amount of dopant that would diffuse across the boundary given the duration and temperature of the step; and the depth to which these dopants would penetrate the lower layer (where the junction would form if the dopant type in the lower layer differs from that of the upper layer).

```
(defprocess Poly-Source-Diffusion
 AKO Ox/Drive-in
 effects
 ((for-all-existing $l \: layer-region (SPI)
   (:IF
    (:AND
     (is-type polysilicon (@ (material $l) (SPI)))
     (:not (is-type polysilicon
                   (@ (material (below $l)) (SPI))))
     (> (* (@ (concentration $l) (SPI))
           (polysilicon-segregation $l))))
        (@ (concentration (below $l)) (SPI))
    (CREATED ($NL layer-region (SPI))
     (member $NL (new-layer(@ (layer (below $l))(SPI)) (PI)))
     (member (@ (layer (below $l)) (SPI)) (new-layers (PI)))
     (change = (below $l) $nl)
     (change = (above (below $l)) $nl)
     (change = (region-type $nl) (@ (region-type $l) (SPI)))
     (change = (top-pos $nl) (@ (top-pos (below $l)) (SPI)))
     (change = (above $nl) $l)
     (change - (concentration $l)
             (/ (xfer-dose $l) (@ (thickness $l) (SPI))))
     (change - (top-pos (below $l)) (junction-depth $l))
     (change - (thickness (below $l)) (junction-depth $l))
     (change = (bottom-pos $nl)
      (- (@ (top-pos (below $l)) (SPI)) (junction-depth $l)))
     (change = (thickness $nl) (junction-depth $l))
     (change = (below $nl) (@ (below $l) (SPI)))
     (change = (dopant $nl) (@ (dopant $l) (SPI)))
     (:IF
      (v= (@ (dopant-type (dopant $l)) (SPI))
          (@ (dopant-type (dopant (below $l))) (SPI)))
      (change = (concentration $nl)
              (* (@ (concentration $l) (EPI))
                 (polysilicon-segregation $l))))
     (:IF
      (:NOT (v= (@ (dopant-type (dopant $l)) (SPI))
                (@ (dopant-type (dopant (below $l))) (SPI))))
      (change = (concentration $nl)
              (- (* (@ (concentration $l) (EPI))
                    (polysilicon-segregation $l))
                 (@ (concentration (below $l)) (EPI))))
     (change = (material $nl)
             (@ (material (below $l)) (SPI)))))))
 constraints
 ((DEFN POLYSILICON-SEGREGATION ($l)
   (segregation-coefficient
    (@ (dopant $l) (SPI)) polysilicon
    (@ (material (below $l)) (SPI))))
  (DEFN XFER-DOSE ($l)
   (transferred-dose
    $duration $temperature $l (@ (below $l) (SPI))))
  (DEFN Junction-Depth ($l)
   (Diffused-Junction-depth
    $duration $temperature $l (@ (below $l) (SPI))))))
```

Figure 4.10 Model of diffusion across polysilicon material boundary

The model implicitly makes a couple of assumptions. First, it assumes that only diffusion between polysilicon layers and the layers below them matter (it assumes no dopants cross from the polysilicon layer to the layer above it). Second, it assumes that the new layer does not completely wipe out the original lower layer: the lower layer is not a thin film that the dopants saturate and cross. These assumptions, and the lack of generality they entail, imply that the model will not correctly predict the outcome in situations that vary slightly in certain ways from the situation in which it is assumed that the model will be used. Thus, as models become less general, they take on more of the undesirable characteristics of experiential causal associations. Experiential causal associations can be thought of as "theoretical" models with an extreme degree of specificity.

The Poly-Source-Diffusion model exhibits how the creation of new layers is handled. The model creates new layer-regions individually. Within the `CREATED` form that creates a new layer-region, two sub-statements are included to manage the collection of these new layer-regions into layers. The first states that the new layer (bound to the variable $NL) is a member of a set of layer-regions associated with the layer of the layer-region below the polysilicon layer. This layer is considered to be the layer that the new layer is "based on." The second statement indicates that this "base" layer is a member of the set `(new-layers (process interval))`. These two statements interact with those inherited from the Create/Modify-Layers model to create the new layer. The relevant statements from the Create/Modify-Layers model are shown in Figure 4.11.

```
(defprocess CREATE/MODIFY-LAYERS
 AKO Modify-Layers
 effects
 ((for-all-true $old-layer e (new-layers (PI))
   (CREATED ($new-layer layer (SPI))
    (for-all-true $lr e (new-layer $old-layer (PI))
     (:AND
      (member $lr (@ (layer-regions $new-layer) (EPI)))
      (change = (layer $lr) $new-layer)))))))
```

Figure 4.11 Definition of Create/Modify-Layers models

The essence of this model is that it creates a new layer for each base layer in the new-layers set, and collects all layer-regions based on each base layer into the set of layer-regions that comprise the new-layer.

## 4.5. Summary

Model-based reasoning techniques confer a degree of robustness on a diagnostic system. Modeling technology is used to explicitly represent the mechanisms underlying a system's behaviour in a general way. This makes it possible to explicitly encode many aspects of the context in which the system operates and how its behaviour depends on this context. Moreover, the effects of changes in the context can be often be predicted. Such contextual changes thus modify, rather than invalidate, the inferences of which the diagnostic system is capable.

We employ a device-centered reductionist approach to modeling semiconductor manufacturing. In this approach, we model the manufacturing operations as parameterized discrete actions, and the manufacturing process as a sequence of these actions. The models describe, in simplified terms, the overall effect of performing each operation on the topology and geometry of the structures created on the wafers being processed. This approach makes it possible to integrate the effects of the entire manufacturing process, keeping track of which operations are causally involved in the creation and modification of which features of the wafer structures.

The only communication between the operations of the manufacturing process is mediated by the structures created on the wafer. Thus, it is necessary to represent these structures explicitly. These structures are complex and three-dimensional. Representing and reasoning about these structures requires reasoning about geometrical shapes and relations, and how these are modified by chemical processes. The ontological choices made in deciding how to represent the structures strongly affects the generality, complexity and utility of the models. In particular, we choose to limit the complexity of geometrical reasoning in our system by limiting ourselves to a simplified representation that tracks geometry in only one dimension and topology in two dimensions.

Features of the semiconductor manufacturing domain complicate the representational machinery needed to reason in this domain. Structures can be created and destroyed, and their number depends on the precise sequence of operations performed. The operations in the manufacturing process are distributed through time. These characteristics imply a need for quantified statements in the description of an operation's effects, and a capability for at least a simple form of nonmonotonic reasoning. To minimize the complexity of reasoning in such an environment, we limit the capability of the reasoner to concrete inferences (forgoing the possibility of abstract

inferences) and employ semantic attachment. Semantic attachment is used both to allow economical algorithms for inference based on iconic representations, and to enable most terms to evaluate to canonical names for the objects they refer to.

The choices we make when deciding how to model a domain limit the range of phenomena that the resulting models can be used to reason about. In particular, our decision to limit ourselves to one-dimensional geometry in the representation of wafer state precludes reasoning about phenomena that hinge on intrinsically two- or three-dimensional effects. Also, our decision not to represent some kinds of operations, such as cleaning steps and photoresist baking steps, precludes reasoning about manufacturing failures that stem from problems with these types of operations.

As we saw in section 4.4.3.2, the decision to represent operations as discrete actions and to represent the distribution of dopant by discrete homogeneous layers makes it difficult to represent the effects of diffusion operations in a completely general way. This is because there is a combinatorial explosion in the number of possible outcomes of a diffusion operation, and a discrete action representation requires that each possible outcome be treated separately. Satisfactory treatment of diffusion operations would require models of behaviour that represent change as incremental evolution, and models of wafer structure that represent the distribution of dopants with greater accuracy. In effect, it would require finer granularity in the representation of both time and space. To limit the complexity of models within our present framework, it is necessary to limit the generality of the models. This implies that when modeling a particular diffusion operation in a manufacturing process, one must make an assumption regarding the state of the wafer at the time of the operation, and choose the model that is most appropriate. Thus, even our representation of theoretical knowledge will contain contextual assumptions.

In conclusion, we note that using model-based techniques to represent theoretical knowledge enables a diagnostic system to be robust in the face of changes in context, because the dependency of the knowledge on such context is explicitly represented. However, the range of phenomena that can be captured is limited. Some phenomena will lie outside the scope of known theory. More importantly, our representational machinery cannot capture some phenomena that are well-understood theoretically, in a manner that admits both succinct representation and tractable

reasoning. Finally, even the representation of theoretical knowledge will include some contextual assumptions.

# Chapter 5. Process Representation

The model-based approach seeks to achieve a certain level of robustness by automatically adapting to changes in the manufacturing plan. It can do so because the manufacturing plan is an explicit input to the system, and all knowledge about the causal relationships in the process is determined by symbolically simulating the process. This implies that it is necessary to supply the system with a representation of the manufacturing plan. Like any other simulator, the DAS qualitative simulator defines its own idiosyncratic syntax for representing manufacturing plans. The first section of this chapter briefly describes that syntax, indicating the information requirements of the simulator.

The models of the actions, and the specifications of the manufacturing plan employ qualitative information primarily. This is supplemented by some quantitative information in the form of analytic equations, and numerical values for some parameters. It was pointed out in section 4.4.1 above that the goal of the simulation is not accurate prediction of the outcome of the manufacturing process, but explication of the causal relationships inherent in it. We assert that to uncover these causal relationships, only qualitative information is required. Further, some phenomena in the domain may not be amenable to accurate quantitative representation by analytic expressions of the sort used by the models. Thus, using primarily qualitative information is both justified and necessary. However, it is useful during the development of the process representation (as well as development of the models) to be able to create a graphic display of the structure of the wafer after each operation. The creation of a graphic display requires quantitative information, because of the necessity to commit to specific coordinates for each wafer-region and layer-region. Accurate values for all parameters and more importantly, accurate analytical expressions for all functions employed in the models, may not be available. Hence, the DAS simulator supports a display subsystem that employs *faux* values for parameters and function definitions. These completely shadow and are kept separate from any true quantitative information provided, to emphasize that they are only intended to facilitate

visualization of the process, and not to reflect an accurate quantitative simulation of it. The second section of this chapter describes this display subsystem.

The fact that every simulator has its own idiosyncratic requirements regarding process representation, and that still other representations are required for running the fabrication facility, leads to a serious maintenance and consistency problem for semiconductor manufacturers. When several representations must each be created by hand, it is very difficult to ensure that all these representations are consistent. Furthermore, as the process is developed and refined, it is difficult to ensure that all the representations in use are kept up to date. One result is that one cannot be certain whether the process being run in the factory is the same as the process that was simulated. Additionally, since each representation is tuned to serve a specific purpose, none of these application-specific representations is sufficiently complete to serve the purpose of documenting the process for technology transfer and corporate memory.

For this reason, research is currently ongoing aimed at developing a unified, computer-readable process representation that can serve all the purposes for which multiple representations are currently needed. The idea is that a single, very rich representation of the process is developed and maintained in a logically centralized *process representation server*. The representations needed for simulators, runsheets, scheduling and other purposes are then automatically derived from this unified representation. The third section of this chapter briefly describes the prototype unified process representation being developed at Stanford in collaboration with Enterprise Integration Technologies, Inc. (EIT), which is embodied in an object-oriented server with persistent storage called the "Distributed Information Service (DIS)" [36]. Finally, a fourth section describes how the information needed to drive the DAS simulator is extracted from that representation.

## 5.1. Representing the Process for DAS

To perform a simulation, the Gordius qualitative symbolic simulator requires three basic types of information about the specific process in addition to the general knowledge described in the previous chapter. As the system is written in Common Lisp, each of these types of information is specified using Lisp function and macro calls.

The first type of information required is a description of the initial *world-state:* a description of the state of the world at the beginning of the simulation. For DAS, this is a

description of the initial state of the wafer that is to be subjected to the manufacturing process. For each structure whose manufacture is to be simulated, one must create a **structure**, a **wafer-region**, a **region-type**, a **layer-region** and a **layer** (assuming that the initial wafer is a homogeneous crystal of silicon). One must indicate the obvious relationships among these, specify the dopant type of the background dopant if any, and if desired, specify values for dopant concentration and geometric dimensions.

A Lisp function called Set-Up-Substrate is provided that will create the initial substrate layer and region-type. Its arguments are: the name to be assigned to the substrate layer (this name with a number appended is given to the initial layer-region of the layer), the name to be assigned to the initial region-type, the name of the substrate material, and the name of the first time-point (usually simply START). An optional argument can be used to specify the species of the background dopant. The **above** and **below** attributes of the initial layer-region are initialized to a constant object of type **ENVIRONMENT** called **THE-ENVIRONMENT**.

Another Lisp function called Set-Up-Structure is provided that will create a structure and the initial wafer-region spanning the width of the structure. Its argument list consists of: the name to be assigned to the structure, the name to be assigned to the wafer-region, the name of the initial region-type (which should already have been created) and the name of the initial time-point. The **left** and **right** attributes of the structure's initial wafer-region are both initialized to be a constant object called **REGION-EDGE**.

Finally, a Lisp function called Initialize-Run is provided that initializes a single structure called **STRUCTURE1** with an initial layer called **SUBSTRATE**, a material called **|Silicon|**, an initial region-type called **LD1**, an initial dopant species of **BORON**, and an initial region called **REGION1**. This function takes only one argument, which should be the name of a function that will construct the descriptions of the masks that will be used by the process.

The second basic type of information required is a description of the topology and geometry of each mask layer within each structure. One must create **mask**s and their constituent **mask-regions**, specify the **opacity** of each mask-region and describe the relative placement of mask-region edges either qualitatively or quantitatively.

To facilitate the construction of mask descriptions, a Lisp function called **BUILD-A-MASK** is provided. The arguments to this function are: the name to be assigned to the mask, the name of the initial time-point, and a list of mask-region descriptions in a special syntax. BUILD-A-MASK assumes that the mask-regions described in the list are adjacent to one another and specified in left-to-right order. Each mask-region description is a list starting with one of the two symbols **OPAQUE** and **TRANSPARENT**. Obviously, the mask-region descriptions will alternate between these two. The remainder of each mask-region description list consists of optional constraints. Each constraint is a list beginning with a keyword specifying the type of the constraint followed by a single argument.

One type of constraint is specified by the keyword **:CALLED.** This constraint permits the mask-region to be named, so that it can be referred to in the descriptions of other masks. All other constraints constrain the horizontal extent and position of the mask region by constraining the positions of the mask-region's edges. The keywords **:FROM**, **:START-BEFORE** and **:START-AFTER** constrain the position of the mask-region's left edge (its **left-pos**), by specifying how the position of the edge relates to the position of other lateral features of either the structure being masked or the other masks. The keywords **:TO**, **:END-BEFORE** and **:END-AFTER** similarly constrain the mask-region's right edge. The list of mask-regions in each mask should span the width of the structure's initial wafer-region. The first mask will be registered most often to the edges of the initial wafer-region, and subsequent masks will be defined in relation to the edges of mask-regions in previous masks. One could also use the **:FROM** and **:TO** constraints to provide quantitative lateral coordinates for the mask edges. Figure 5.1 illustrates a simple mask-description that defines mask edge positions relative to the edges of both the initial wafer-region (**REGION1**) and mask-regions of a previous mask (**COLLECTOR-MASK**).

It is necessary to give each mask level for each structure a unique name. Further, any names given to mask-regions must be unique across all masks for all structures. A generic-function called **MASK-LEVEL** is used to associate the appropriate mask with each tuple of mask-level and structure.

```
(build-a-mask 'ISOLATION-MASK 'START
 '((opaque (:CALLED ISOLATION-MASK-LEFT)
           (:FROM (left-pos REGION1))
           (:TO (right-pos COLLECTOR-MASK-LEFT)))
   (transparent (:CALLED ISOLATION-MASK-TRANSPARENT1))
   (opaque (:CALLED ISOLATION-MASK-MIDDLE))
   (transparent (:CALLED ISOLATION-MASK-TRANSPARENT2))
   (opaque (:CALLED ISOLATION-MASK-RIGHT)
           (:FROM (left-pos COLLECTOR-MASK-RIGHT))
           (:TO (right-pos REGION1))))
```

Figure 5.1 Simple example of a mask description.

The third basic type of information required for the simulation of the process is a list of the processing operations comprising the manufacturing plan. Each operation is expected to be an instance of one of the actions modeled. A Lisp macro called **FAB-PROCESS-INSTANTIATE** is used to create each instance in the list. This macro takes one required argument, the name of the action to be performed, and any number of optional constraints. Generally, the constraints are statements in the same lisp-syntax logic used in the models of the actions. However, some constraints need to be specified with such regularity that special syntax is provided to facilitate their inclusion.

One set of constraints that should be included in the description of every action concerns the time-interval during which the operation takes place. DAS must be told explicitly how the time-intervals for each action relate to one another. Also, it is generally useful to give the time-interval of each action a meaningful name. A special syntactic form is provided that permits several constraints to be specified succinctly in a single statement. The form of this statement is:

```
(INTERVAL <preceding> < <interval-name> < <following>)
```

where <preceding> and <following> are specifications of time-points. This single statement indicates that the name of the time interval is <interval-name>, and that it occurs after <preceding> and before <following>.

Since most of the action models are parameterized, it is necessary to specify or somehow describe the values of these parameters. One way to do this is to include a constraint statement of the form:

```
(called <parameter name> <parameter value>)
```

Such a statement allows either a quantitative or symbolic value to be specified for the parameter. It should be noted that providing quantitative values for numerical parameters may not be sufficient to constraint the simulation to a unique outcome. They will only be sufficient if arithmetic expressions or semantically attached lisp functions are provided for all relevant functions mentioned in the action models. The alternative approach is to write a statement that qualitatively constrains the value of the parameter(s) by partially indicating the expected outcome of the operation. For example, one might specify that the duration and temperature of an oxidation step are sufficient to grow an oxide of a given thickness. Statements of this type not only help to constrain the simulation, but also partially document the intention of the process designer. Statements of this kind can be very weak, as long as they unambiguously define the outcome of the action. For example, the constraint for the oxidation step may merely indicate that the duration of the operation is not sufficient to completely oxidize the entire substrate layer.

These qualitative constraints will generally require universal quantification, because at the time the constraint is being written the number of individuals affected by the constraint may not be known, and they are likely to be also to be anonymous. That is, as processing proceeds new individuals, such as new layer-regions, are created. These individuals will have system-assigned names, and their number will depend on how the actions of the process interact. To facilitate the writing of such constraints, a predicate called **NAMED** is provided. This predicate permits the process designer to provide meaningful names for individuals that are expected to be created during an action, so that these may be referred to by name in constraints associated with subsequent actions.

Figure 5.2 displays a simple example of how an oxidation step might be specified. In the example, the time-interval is given the name **STEP1**. This interval occurs after **START**, which is the name of the initial time-point of the simulation. The example illustrates the use of a time-point called ***NOW***. The ***NOW*** time-point merely serves to indicate that the **STEP1** time-interval is finite. It is not necessary to indicate that it ends before the beginning of the next step, as the description of the next step will indicate that that step begins after the end of the current step. During a simulation, if the execution is interrupted after any given step, the user can conveniently use the ***NOW*** time-point to refer to the latest values of all time-dependent attributes.

The operation described in figure 5.2 includes two additional constraints. The first indicates that none of the layer-regions in the **SUBSTRATE** layer are completely oxidized. The second indicates that any new layers created (only one is expected) should be named **INITIAL-OXIDE**.

```
(fab-process-instantiate OXIDATION
 (interval start < STEP1 < *now*)
 (for-all-true $lr e (@ (layer-regions SUBSTRATE)
                         (start-of STEP1))
  (< $duration
     (oxidation-destroy-time $lr
                              $temperature (start-of STEP1))))
 (for-all-existing $l \: layer (end-of STEP1)
  (:=> (created-during $l STEP1)
       (named $l INITIAL-OXIDE))))
```

Figure 5.2 Instance of an Oxidation operation

## 5.2. Representing the Process for Display

The qualitative simulation process is a highly abstract computation that involves forming literally thousands of formulas and determining for each whether they are true or false. How can a user reassure his or herself that the simulation is proceeding correctly, that the process specification, for example, is correct and sufficiently unambiguous? Two standard techniques are to generate a log of the progress of the simulation, and to provide ways to visualize it.

The DAS system generates a simple log indicating which step is being executed, when it finishes, how long it took and how much time the entire simulation is accumulating. The steps are identified by the name given to the time interval by the **INTERVAL** constraint described in the previous section. The user can augment this identification by providing a descriptive string to be displayed in the log. This is done by adding to the process specification a special constraint of the form:

```
(display-message <descriptive string>).
```

For semiconductor manufacturing, the obvious visualization tool to provide is one that creates a display of the structures being created on the wafer similar to the drawing in figure 4.4 of the last chapter. However, this poses a difficulty. To create such a display requires that numerical coordinates be determined for corners of each **layer-region** displayed. The quantitative information required to determine such coordinates will often be lacking for two reasons. First, quantitative values for the parameters to each

model may be missing. During the early phases of design, one can use the DAS simulator to qualitatively simulate the process design before such quantitative information has been determined.

More importantly for our case, which is concerned more with diagnosis than design, even when quantitative values are established for all real-valued parameters, the simulator may not be able to determine quantitative values for the wafer structure attributes that are determined by those parameters. The simulator can propagate quantitative information through the analytic expressions in its models, whether that information consists of unique values or interval bounds on the values. However, the models will often employ functions for which no analytic expressions are known to the simulator. This is due to the fact that the phenomena being modeled are not amenable to accurate representation in analytic expressions. In these cases, the models employ functions so as to indicate that a functional relationship exists between the value of the function and the quantities given as arguments to the function. However, the models do not try to express the relationship mathematically in terms of an analytic expression, because no such analytic expression could accurately represent the relationship.

It is possible in many cases to provide analytic expressions for many of the functional relationships that apply within a specific narrow range of operation. The expressions employ "tuning" parameters, and are essentially curves fitted to empirical data. This is the approach used in the Fabrics statistical simulator [37]. The focus of that work was to provide a very fast quantitative simulator that could be used in Monte Carlo simulations to characterize the behaviour of a process in the neighbourhood of the operating point for which the analytic expressions were tuned. Even in the Fabrics work, however, it was found that diffusion phenomena required numerical techniques because analytic expressions were not sufficiently accurate.

Furthermore, the goal of the DAS simulator is not accurate quantitative simulation. As was stressed in the previous chapter, the goal is to elucidate the causal and functional relationships in the manufacturing process. In some cases, the curve-fitting analytic expressions do not lend themselves easily to incorporation in a model organized around the process engineer's conceptual understanding of what is happening in the process.

Finally, there is a pragmatic reason for avoiding the incorporation of such analytic expressions. Process engineers have a great need for accurate quantitative simulators, and that is what they have historically been offered. For this reason, once it is possible to simulate the process quantitatively, improving the accuracy and precision of the simulation becomes an overriding focus and priority, and lack of accuracy is seen as a failing of the simulator and its models[3]. Keeping the DAS simulation strictly in the qualitative domain helps to stress that DAS is not competing with numerical simulators in the accurate quantitative prediction of the results of processing, but attempting to serve a completely different purpose.

We desire a display that is qualitatively rather than quantitatively correct, that reflects as much as possible what the simulator has determined to be true about the structure of the wafer. To accomplish this, the DAS system determines symbolic expressions for the geometric attributes of the **layer-region**s by tracing the dependencies through the wafer history. The resulting symbolic expressions involve only the parameters to the models of each step, and those functions that are not further defined within the models. The expressions are then evaluated to determine the coordinates needed for the visual display, using approximate values for the parameters and Lisp functions that very roughly approximate the undefined functions. These approximate values and functions constitute a *faux* quantitative simulation that is parallel and independent of any quantitative reasoning done within the qualitative simulator.

By keeping these separate from the models and the parameter values provided in the process specification, it is emphasized that their only purpose is to support visualization, and that they can be left out entirely if visual displays are not desired. In particular, it is emphasized that the faux Lisp functions do not attempt to achieve numerically accurate estimates of the functions employed in the models. Rather, they provide very simple, rough approximations suitable only for producing a reasonable-looking diagram.

---

[3]This was the experience with the AI Process Simulator at Texas Instruments.(Robert Hartzell, personal communication).

To provide the faux values for parameters, the process specification can include constraints of the form:

```
(display-parameters (<name1> <value1>)…(<nameN> <valueN>))
```

where <name1>…<nameN> are names of parameters and <value1>…<valueN> are corresponding numerical values for the parameters.

The function **fab-display-fn**, if provided to the simulator as an argument at the beginning of the simulation, will be used by the simulator to generate a display of the wafer structure at the end of each step. Since the simulator creates a wafer history that encodes the entire history of the structure of the wafer, the function also can be used to view the wafer structure at any quiescent time-point that has already been simulated (generally, the **start-of** or **end-of** any interval that has been simulated).

The display reflects the information known to the qualitative simulator because it is based on symbolic expressions extracted from the wafer history produced by the simulator. However, it should be noted that one should take considerable care in interpreting the display. Some features in the display may be accidents produced by the choice of faux parameter values and functions. For example, two **layer-region**s that appear to have the same height on the diagram may not be known to have the same height by the qualitative simulator. To check what the simulator actually knows, it is necessary to evaluate sentences in the simulator's logic to determine what the simulator knows about their truth. This is facilitated by a lisp function called **term-eval-print**.

When the form **(term-eval-print)** is evaluated in a Lisp Listener window, a new reader is invoked that reads formulas in the modeling language and evaluates them. If the formula is a sentence, its truth value is printed (actually, the TMS node indicating its truth value is printed). If the formula is a term, then the value of the term in the semantically attached Lisp interpretation, if any, is printed. Furthermore, after a sentence is evaluated, one can enter the symbol **why**. This invokes a simple explanation system that facilitates determining why the system (dis)believes the sentence, by listing the sentences that support the original sentence and their truth values. These sentences are numbered, and one can recursively question why particular sentences are (dis)believed by entering the corresponding number.

## 5.3. The DIS Semiconductor Process Representation Server

There are many different activities throughout the life cycle of a manufacturing process that require some representation of exactly what the manufacturing process is. In addition to the “runsheets” needed to actually run the manufacturing process in the factory, other representations of the process are needed to drive technology simulators to aid in designing, improving and debugging the process, as well as to drive factory simulators and schedulers to aid in optimizing cost, resource utilization, product throughput and other variables.

Each of these applications employs a different model of semiconductor processing and processing plans, and each requires its own idiosyncratic representation of the manufacturing process. The need to create and maintain multiple redundant representations incurs a high cost in duplication of effort. As well, it is very difficult to ensure that all these ad hoc representations are kept in synchrony as the process is developed and modified. Furthermore, none of these specialized representations is a sufficiently complete description of the process to serve the functions of technology transfer and corporate memory.

To remedy these problems, several organizations encompassing both industry and academia are developing the notion of a comprehensive “Semiconductor Process Representation (SPR).” The intention is that only a single representation of the process be developed and maintained, sufficiently rich to serve all the purposes that currently require multiple ad hoc representations. In order to drive computer simulations, and also to enable automation in the factory itself, this representation must take a machine readable and manipulatable form.

The fact that manufacturing plans can be viewed as procedures for carrying out the manufacturing has led some researchers to investigate the use of a programming language approach to this representation problem [38-41]. However, manufacturing plans are more than just procedures. Also, more information than just the manufacturing plan itself (such as information about equipment and facilities, for example) will eventually have to be represented in a fashion that permits a rich interconnection with the representation of the manufacturing plan. Finally, the fact that different applications model semiconductor manufacturing differently may imply a need for the ability to reason about the structure of the plan, and suggests that the representation of that structure should be more declarative than is generally the case in

programming languages. These and other reasons have led to investigation of declarative, object-based approaches to the problem [42-48].

The Distributed Information Service (DIS) [36] is an object-oriented system for sharing knowledge with persistent storage in a client/server environment. It was developed to be a logically central (though possibly physically distributed) electronic reservoir of information concerning all aspects of semiconductor design and manufacturing. It permits client applications to interact with it at two different levels of abstraction: the “Abstract Object Interface (AOI)” defines an abstract object-oriented framework for representing any kind of information, that features dynamic class creation, dynamic reparenting (reclassification), and dynamic, multiple inheritance; the “Domain Specific Language (DSL)” provides abstractions that correspond to important concepts in a specific domain, and hides the implementation of these concepts in the underlying AOI framework.

At the AOI level, applications interact with DIS in terms of OBJECTs, their SLOTs, FACETs on those SLOTs, and METHODs written in a Scheme-like extension language. The objects are identified by name, and exist in different NAMESPACEs. The OBJECTs are organized into an inheritance hierarchy, or taxonomy (actually an inheritance graph, as multiple inheritance is supported). The AOI level makes no commitments regarding the ontology used to represent information: applications are free to invent their own ontologies.

At the DSL level, interactions are in terms of concepts in an extensible, domain-specific ontology to be shared by all applications, so that information-sharing is enhanced. Although the DIS system is being used to represent information of many different kinds useful for device design, facility management, and process design, semiconductor process representation is currently the only domain for which a DSL has been defined. The basic concepts in this DSL include STEPs, EQUIPMENT, PARAMETERs, ATTRIBUTEs and DOMAINs. STEPs can be hierarchically composed of other STEPs; STEPs and EQUIPMENT can have PARAMETERs; PARAMETERs can have ATTRIBUTEs, and can belong to DOMAINs. STEPs and EQUIPMENT also participate in an inheritance hierarchy (taxonomy). Extension of the ontology is through the creation of specializations within the taxonomy, to represent specialized classes of STEPs and EQUIPMENT (for example, to differentiate etch steps from diffusion steps). The first-level partition of the STEP concept is into TOP-LEVEL-PROCESSes (intended

to represent “complete processes”), PROCESS-MODULEs (intermediate aggregations), PROCESS-STEPs (roughly corresponding to the ill-defined but ubiquitous concept of a “unit process”) and EQUIPMENT-OPERATIONs.

In addition to these two levels of abstraction, DIS provides operations for session management, transaction management, client authentication and programmable notifications to interested clients of when changes are made to particular objects in DIS. Ongoing research is aimed at adding design configuration management and more sophisticated access control.

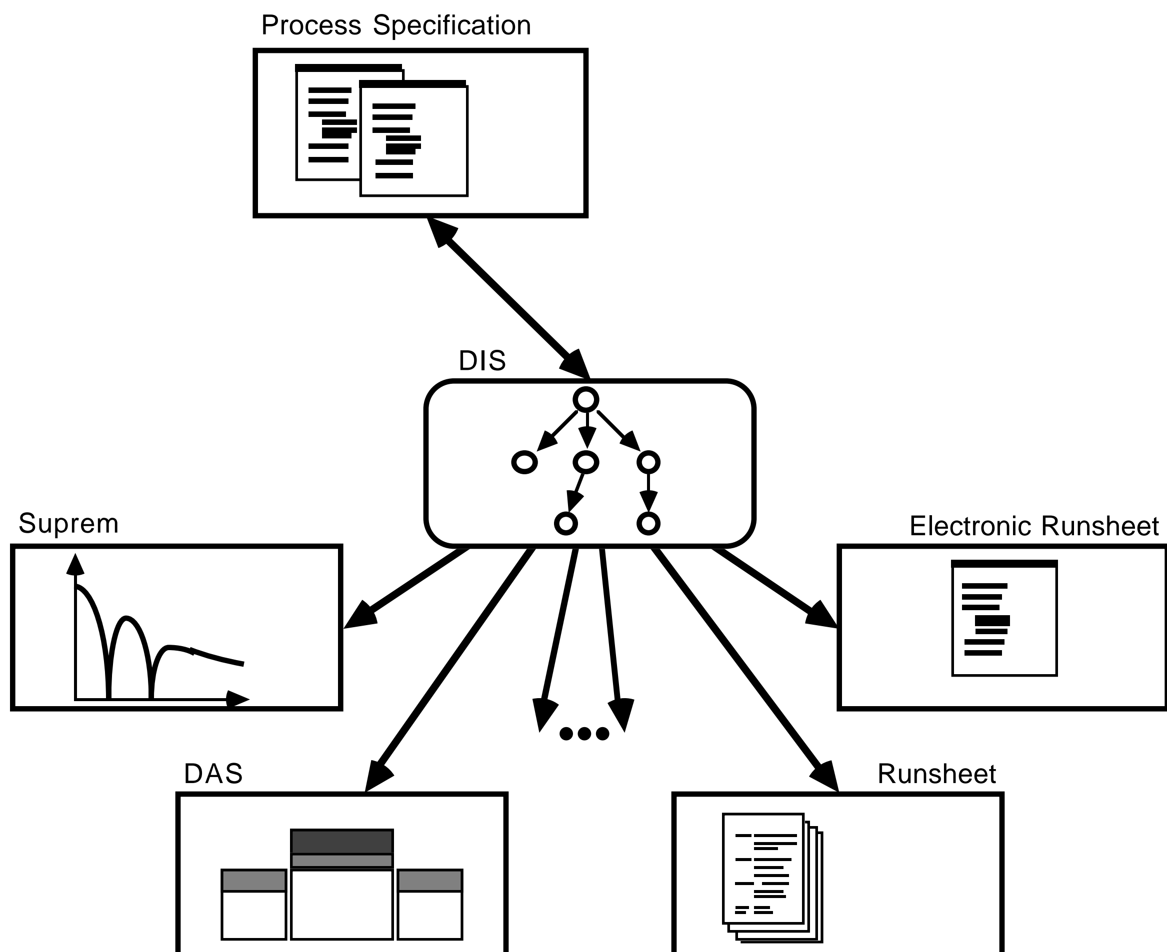


Figure 5.3 Semiconductor Process Representation Server scenario

Communication with the server is through a (programming language independent) protocol using character-strings over a stream-based network protocol (TCP/IP). However, special support is provided for $C^{++}$ applications in the form of a

client library that provides an object-oriented interface that hides the string-based protocol.

Figure 5.3 illustrates the scenario for use of the DIS system as a semiconductor process representation server. A process specification system is used to compose the process plan and store it in the DIS server, incorporating as much as possible pre - existing steps from a public section of the inheritance hierarchy that serves as a “component library” for semiconductor processes. Then, applications that require information about the process plan , such as simulators and runsheet generators, obtain the information by accessing the server.

## 5.4. Extracting the DAS representation from DIS

Given that a rich representation of the manufacturing process is stored in the DIS server, the question arises as to how it can be used to drive the DAS qualitative simulator. Quite apart from the low-level issues of communicating with the server over the network and syntactically transforming the information obtained into the format required by the simulator, there are two somewhat thornier issues. These issues arise not only for DAS, but for any existing application that must access the information. Future applications may be designed with the current process representation in mind, thus alleviating these issues somewhat. However, they will probably continue to arise even for such “process representation-aware” applications.

The first issue concerns the need to translate the representation of steps and their parameters in the process representation into the conceptual vocabulary of the various applications. The process representation in the server necessarily makes ontological commitments concerning what kinds of operations occur in a process, and what information is available about each operation. These commitments, the “language” used to describe processes will not, in general, match the commitments made by the designers of different simulators and other applications.

The second issue concerns the fact that different applications have different views regarding what is to be considered an atomic operation (a “step”) versus a composite operation (a “module”). The process representation in the server will have made choices about the temporal boundaries of operations, and the compositional granularity of the representation. Once again, these may not coincide with the way other applications choose to make these decisions.

We consider these two issues in turn in the next two subsections. Then we describe how we would prefer that model selection be handled, and finally, how pragmatic considerations force us to fall short of this goal.

### 5.4.1 Step Taxonomy and Model Selection

Different applications “think about” semiconductor processing differently, and no single application needs all the information that all of them require collectively—therefore some data reduction and translation is required. One way of looking at this problem is to think of it in terms of determining how the steps in the representation of the process are to be *modeled* for each application.

Each application defines its own vocabulary for describing what goes on in the fab. Writing a specification, an “input deck” for SUPREM, for example, involves mapping the activities that take place in the fab into the conceptual categories of processing operations understood by the simulator, and its vocabulary for them.

The DIS representation permits steps to be organized into a taxonomy. The taxonomy groups steps that have a common abstraction into classes. There are many different ways to organize manufacturing operations into abstract classes. Indeed, different users likely have different ideas about what the organizing principle in the taxonomy should be.

We are not restricted to one such organizing principle: the representational machinery allows for multiple-parent taxonomies, so that a given step can be situated simultaneously within different abstract classes. Also, it is possible and sometimes useful to use different abstraction principles at different levels in the taxonomy. Here we will explore the relationship between the organizing principle of the taxonomy, and the problem of selecting appropriate models for simulating a process.

The primary constituents in the description of a step are PARAMETERS. There are at least three different categories of parameters. To support runsheet generation and to drive the ‘programmable factory’ as well as to support equipment simulators, there must be SETTINGS parameters that describe what the process equipment settings are for the operation. Since such settings information is apt to be specific to the type of equipment used, it can only appear in the description of steps that are sufficiently specific and concrete that at least the type of the equipment has been assigned.

Process simulations employ models which often operate at a more abstract level. To support these simulations, two additional categories of parameters are required: EFFECTS and ENVIRONMENT. The effects parameters describe the impact of performing the operation. For a chemical vapor deposition step the effects parameters might include the thickness of the layer deposited, and its chemical composition or material type. Such information helps to document the intentions of the process designer in employing the step, and is needed to drive some simulators that are not physically-based.

The environment parameters describe the micro-environment created at the surface of the wafer. These allow physically-based simulators to determine the impact of performing the operation by determining what physical and chemical processes will occur.

In well-designed equipment, one should expect the distinction between these categories of parameters to blur: the purpose of the equipment is to achieve the desired effects by manipulating the wafer's environment. Thus, one will often find that the equipment's settings are directly calibrated in terms of environmental or (less frequently) effects parameters. For example, one of the key environmental parameters in a furnace is temperature. Through the use of continuous feedback control, this can also be an equipment setting, and the power delivered to the RF coils or heating lamps can be an internal variable hidden from the user. As another example, a ion implanter can have a setting that directly specifies the dose (an effects parameter), because it can accurately count the number of ions being implanted per unit time. Nonetheless, it is necessary to have environment and effects parameters, because equipment is not always so well-designed (suitable feedback mechanisms may not be available) and because it is often desirable to simulate a process at an abstract level before equipment has been assigned and settings parameters have been determined.

Given the nature of this representation of the process, there are two sources of information concerning how each step should be modeled for simulation: the types of parameters associated with the step and their values, and the abstract classes into which the step has been placed in the taxonomy.

The CAFE system developed at MIT [45] attempts to identify the correct way to model each step by looking only at the step's parameters. For example, if the step

description includes a temperature environment parameter with a high value, and an ambient-gases parameter that indicates the presence of an oxidizing agent, then the system can conclude that the step should be modeled as an oxidation/diffusion step. This approach requires a fairly high degree of sophistication in the model selection process. The model-selector is essentially an expert system that 'understands' enough about the simulator and its models, and about semiconductor processing in general, to automate the conceptual mapping between them.

An alternative approach is to rely primarily on the abstract classifications of the step to indicate the appropriate model. This approach is computationally much simpler: rather than trying to capture the expert's knowledge about how steps can be modeled within different simulators, it merely provides a mechanism by which the expert can directly specify what model to use for each step by specifying what abstract classes the step belongs to. Model information is associated with the abstract classes of steps, and inherited to the specific steps in the process representation. Although this approach avoids the problem of encoding the "first principles" knowledge of processes and models, it imposes constraints on the form of the step taxonomy. The abstract classes in the taxonomy have to correspond to the abstractions made by the simulator models.

It should be noted that the problem of determining how to model an operation for a given purpose can be viewed as a classification problem: one wishes to know how the operation should be classified within the conceptual framework of the given application. Indeed, one way to perform description-based model-selection would be to create a KL-ONE [49] type classification taxonomy for the operations understood and modeled by the application, and use the description to classify the operation relative to that taxonomy. The alternative approach merely prefers to allow this classification to be done by hand by the experts who best understand what the operation is intended to accomplish and how the applications work.

These two approaches can be combined in a simple fashion. A classification scheme can be used to determine the appropriate model for a given simulator except in those cases where manually-specified model information is inherited from an abstract class. Also, the model information inherited from an abstract class can be ambiguous, representing several models, and a classification scheme can be used to determine the most appropriate model among them based on the values of the parameters.

What abstraction principle provides step classes suitable for the attachment of simulation models? In other words, what kinds of abstractions do simulator models make? Three principles can be identified. These correspond to the parameter categories discussed above.

Wafer-State Transformation: Some simulator models ignore equipment specifics and physical mechanisms, describing operations directly in terms of the nature of the induced transformation to wafer-state. These correspond to (and are often driven by) effects parameters. Such models group together steps that have similar effects on wafer structure, independently of how such effects are achieved. An example of this might be the SUPREM model for Etching operations. Independently of whether a wet etch, plasma etch, reactive ion etch or ion milling step is employed, the impact of the etch step is to partially or completely remove a layer of a given material at the surface of the wafer.

Physical Mechanism: Many simulator models distinguish steps on the basis of the dominant physical or chemical process by which the steps achieve their effects. These models are primarily driven by environment parameters, though effects parameters may also play a role. The SUPREM Diffusion model is an example of this. Most process steps that involve high temperatures, and thus involve diffusion of impurities, are modeled by this parameterized class.

Equipment: Finally, some operations involve wafer micro-environments that are not well-understood or easily described. Such operations are often simulated by models that are equipment-specific and driven by settings parameters, or other equipment-specific information, such as parameters describing ***machine-state***.

Wafer-State Transformation models are more abstract than Physical Mechanism models, which are more abstract than Equipment models. This suggests that it may be possible to construct a single taxonomy that employs all three principles in turn. Near the root of the taxonomy, the classes would be distinguished on the basis of how Wafer-State Transformation is described. Further down in the hierarchy, the more specialized classes of steps would distinguish sub-classes on the basis of the Physical Mechanism (how the microenvironment at the surface of the wafer is described). Closer to the leaves of the taxonomy, steps would be distinguished on the basis of Equipment specifics.

Figure 5.4 below shows a small piece of a potential taxonomy of processing steps that follows this principle. The Etch step merely indicates what parameters are used to describe the effect of performing an etch step: the (possibly selective) removal of material from the surface of the wafer. Its immediate specializations distinguish between two major physical mechanisms used to accomplish etching, namely Wet Etch and Dry Etch techniques. Finally, the most specialized objects in the hierarchy as shown indicate how plasma etches on specific classes of equipment (AMT and Drytek plasma etchers) are described.

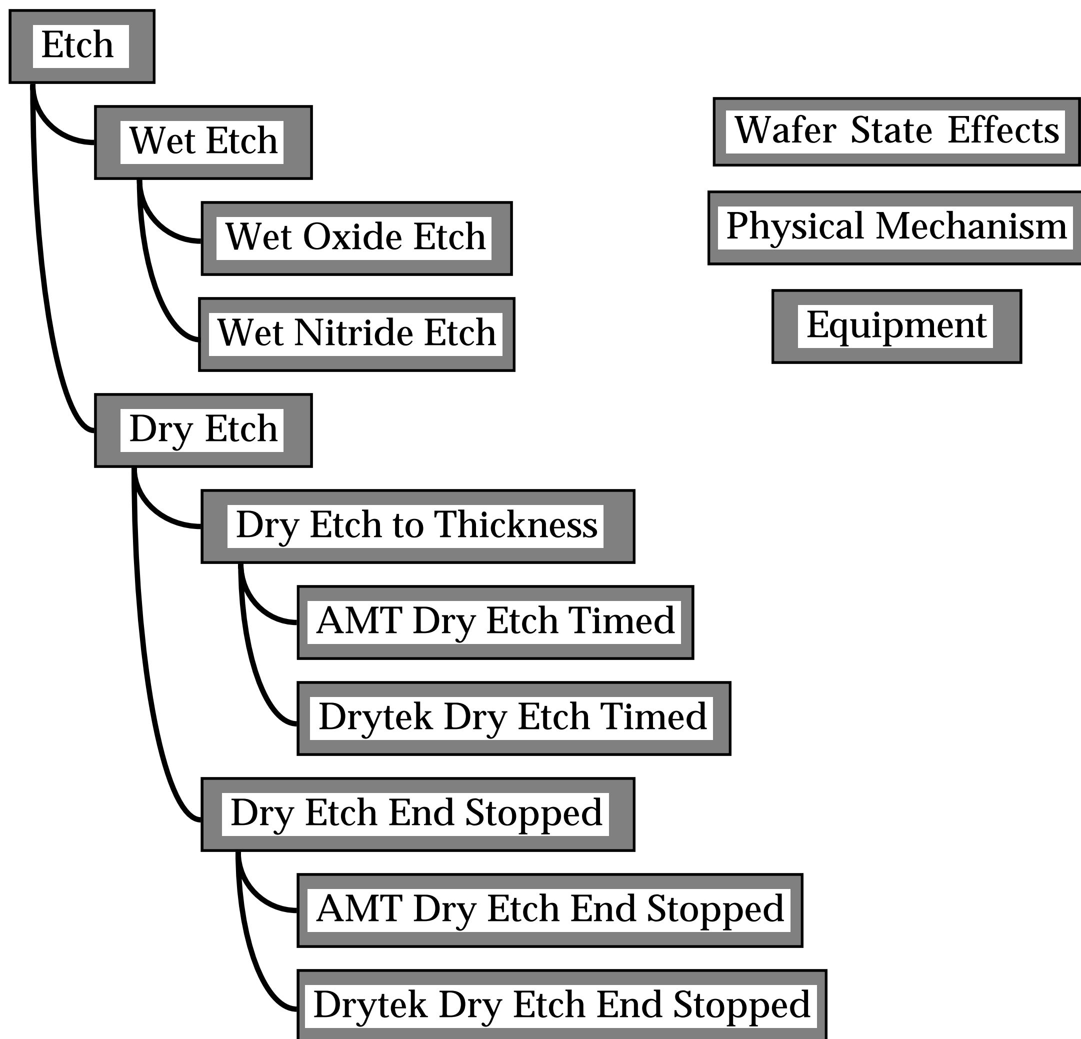


Figure 5.4 Partial Step Taxonomy

The DAS model of the etch operation is driven partly by effects information and partly by environment information. Further, it can operate without any environment information, if the effects are sufficiently specified. Thus, by attaching the general Etch

model of DAS to the high-level ETCH step in the process representation taxonomy, it would be inherited to every etch step in the taxonomy. More specific versions of the DAS etch model, such as END-DETECT-ETCH, could be attached to suitable objects slightly lower in the taxonomy, overriding the general etch model (but of course, incorporating it through DAS' own notion of inheritance of model descriptions).

### 5.4.2 Process Decomposition and Model Selection

In addition to organizing steps into a taxonomy, the representation also allows for the hierarchical composition of steps into ever larger modules, until at the root a single object represents the entire process.

The diagram in figure 5.5. represents what the initial level of decomposition of a process might look like.[4]

Near the top of the composition hierarchy, the modules will represent quite complex sequences of procedures, and it will make most sense to base their taxonomic classification on the purpose they serve relative to the overall goals of the process. This is indicated by the kinds of names I've chosen for these modules.

Closer to the bottom of the composition hierarchy, however, things get a little messier. Objects which are clearly modules, in the sense that they are composed of other steps, begin to enter the realm of what might be called steps from a particular viewpoint.

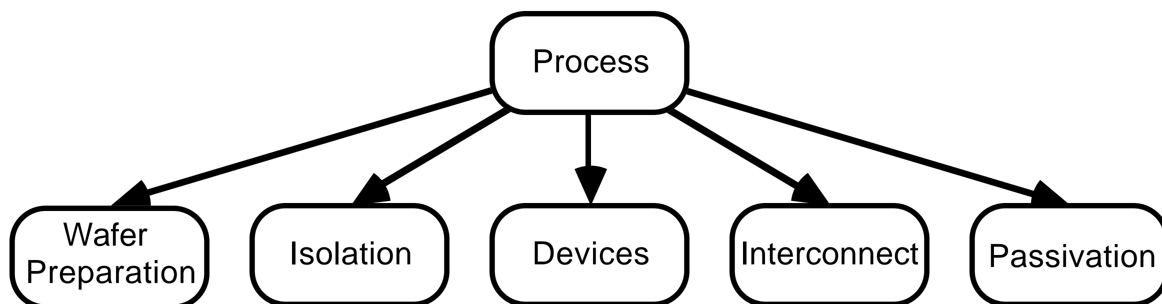


Figure 5.5 Initial decomposition of semiconductor process plan

---

[4] I am indebted to Wes Lucasek (personal communication) for this particular top-level decomposition.

For example, consider the simple situation in figure 5.6. We have a module consisting of an oxidation step followed by an inspection step. From the perspective of simulators and for other reasons, it makes sense to consider the oxidation and the inspection as two separate steps. After all, the inspections are not simulated. On the other hand, when it comes to generating a run-sheet, or driving the factory with an electronic runsheet, there are good reasons for considering the combination of the oxidation and inspection operations as a single step, because the measurements made during the inspection (and the instructions for performing them) are to be closely connected with the operation that made the structure being measured.

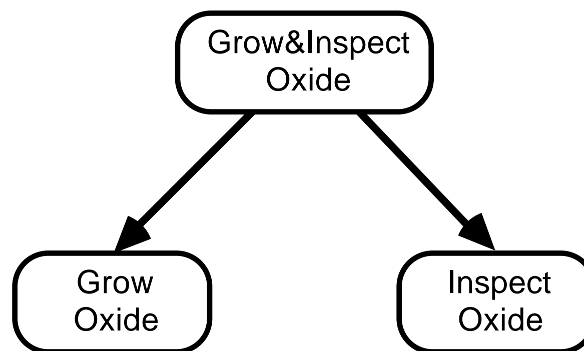

Figure 5.6 CIM versus TCAD notion of “step”

Thus, what is a module from one perspective is a step from another.

This notion is driven home by the more complicated example in figure 5.7, which I’ve simplified by removing all inspections, rework loops, etc. Which are the steps, which are the modules, and which are the substeps?

Taking the CIM view, we might decide to classify the objects that have shading behind them in the figure as steps. Notice that some objects classified as steps are composed of other step-like objects, which might be called substeps. Thus, some of these steps behave somewhat like modules. For the purpose of generating a runsheet, one might go further, and classify the “Pattern Resist” operation as a “step.” This is because it is frequently the case that the same photolithography procedure is used for every mask, and so it is fully described only once at the beginning of the runsheet. Then, instances of the procedure describe only how they differ from the generic procedure. The initial description of the photolithography procedure is parameterized, and the instances of it in the process specify the values for those parameters.

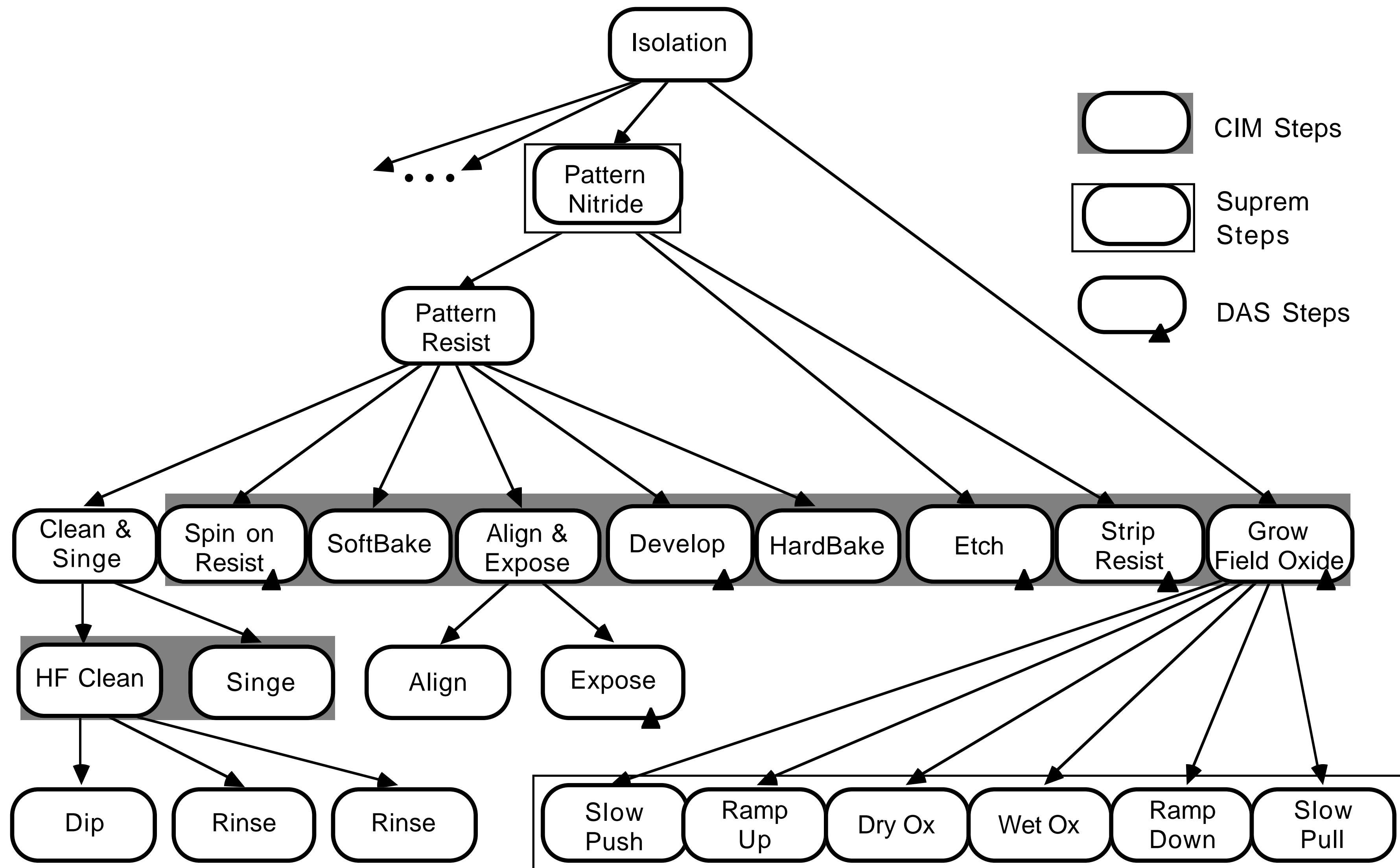


Figure 5.7: Breakdown of part of Isolation module, showing parts that might be classified as steps from different perspectives.

The objects framed by border lines in figure 5.7 indicate operations that might be considered steps by the SUPREM simulator. SUPREM does not explicitly model the photolithography operations of coating the wafer with resist, exposing and developing the resist, and stripping it off. Instead, it has the concept of a patterned-etch, in which the impact of an etch operation is magically confined to those areas of the wafer that would be unprotected by resist. Thus, for SUPREM, the “Pattern Nitride” module would be a step: as the application driving the simulation traversed the composition hierarchy, it would want to stop at the level of the Pattern Nitride step and simulate that, without bothering to look at the steps that the module is composed of. This means that like steps, some modules must have parameters specifying effects, environment, settings, and so on (in this case only effect parameters are needed). Also, we must be able to associate modeling information with modules just as we do with steps, and in fact we desire that these modules have a place in the step taxonomy so that they can inherit model information as appropriate.

SUPREM can be used to model the oxidation step as a single step, or it can be used to model the substeps individually, for greater accuracy. Thus, what we’ve said about steps and modules also applies to sub-steps.

Finally, those objects containing small black triangles indicate how DAS would view the situation. We show this example just to emphasize that not only do simulation and CIM views differ, but there will also be differences among simulators and among different ways of using simulators.

If each application made different decisions regarding where module boundaries lay, then it would be very difficult to extract a reasonable abstraction of the process for the application from the central representation. The rationale used by each application for aggregating steps and modules into larger modules would have to be codified so that this aggregation could be carried out automatically. However, thus far it has been our experience that, although different applications have different views regarding what constitutes an atomic operation, it is possible to construct a single system of aggregations that satisfies all applications. With a sufficiently detailed and carefully designed system of aggregations, the applications only disagree as to the granularity of the representation, and each can find

aggregates in the composition hierarchy that correspond to their individual notions of atomic operations.

The fact that different applications will treat modules at different aggregation levels as steps is an argument in favour of a uniform mechanism for composition that does not distinguish between modules, steps and substeps. Rather it should allow this distinction to be made by each application according to its own need. In particular, modules should be capable of having descriptive parameters, and should participate in the same taxonomic hierarchy as steps, so that they can inherit modeling information. Moreover, it should always be possible to refine the description of a “step,” by indicating that it can be further decomposed.

### 5.4.3 A Simple Approach to Model Selection

In this subsection we assume that steps are organized into a taxonomic hierarchy according to a succession of principles: Wafer-State Effects → Physical Mechanism → Equipment Specifics, and that the distinction between modules, steps and substeps is left to individual applications to make. We describe a simple, flexible mechanism by which appropriate models for process steps can be selected for each application.

We illustrate the mechanism by indicating how a specific application, the DAS qualitative simulator, would be added to the repertoire of applications supported by the process representation.

The first step is to add a new parameter to STEP objects. The name of the parameter is immaterial, as the parameter will be accessed solely by the domains to which it belongs. However, a name appropriate to the use of the parameter would consist of the name of the application concatenated with the word MODEL, separated by a hyphen. In our case the parameter could be called DAS-MODEL. It may be necessary or convenient to be able to attach more than one such parameter to the same step object, in which case the names might be distinguished by a concatenated number. These parameters are indicated as belonging in the domains DAS and MODEL, and perhaps also others, such as SIMULATOR. The default value of any such parameter indicated on the most general STEP object is NULL, which value is interpreted as specifying that no model applies.

Next, a taxonomy of MODEL-OBJECTs (in our case DAS-MODEL-OBJECTs) is created. This taxonomy contains objects representing every DAS process step model. The organizing principle for this taxonomy derives from the most appropriate abstraction principles applicable to models of the application. For any simulator, this may parallel somewhat the abstraction principles used to organize the taxonomy of process step objects. However, it will not likely duplicate it completely. For the DAS simulator, the taxonomy would reflect the organization indicated in figure 4.5 of the previous chapter.

It is intended that the objects in this MODEL-OBJECT taxonomy are suitable values for the <application>-MODEL (e.g. DAS-MODEL) parameters in the step objects. An abstract (non-leaf) object in this taxonomy would be assigned as the value of the model parameter when it is desirable to indicate that the choice of model is ambiguous. In such a situation, the specializations of the model-object should represent possible resolutions of the ambiguity.

Another way to indicate an ambiguous choice of model is to have more than one model parameter for the same application on the step object for which ambiguous assignment is desired, with each such parameter indicating a possible resolution of the ambiguity. In either case, the idea is to allow a parameterized class of process steps to have an ambiguous model assignment. On specializations of these steps, which inherit the ambiguous model assignment, the ambiguity is resolved by examining the values of the step's parameters that distinguish the specific subclass to modeled.

The value(s) of the model-parameter(s) is(are) specified for each step object. Hopefully, the organization of the step object taxonomy is such that it is only necessary to make this manual specification for a small number of abstract step objects. Those values can then be inherited automatically to more specialized step objects.

Next, it is necessary to indicate which parameters on step objects are pertinent to the application that is attempting to model the step. This is accomplished by having a domain name specific to each application, and indicating that the pertinent parameters belong to that domain. In our case, the parameters on step objects that contain information needed by the DAS simulator would be indicated as belonging to the DAS domain.

The precise content of the application MODEL-OBJECTs is a matter of taste. The MODEL-OBJECT should contain sufficient information to enable the application to process any step for which the model-object is the model-parameter value. The judgment to be made concerns how much information can/should be declaratively represented in the SPR server (and thus made accessible to other applications), versus how much must/should be represented only in the application itself.

In the case of the DAS simulator, the models are qualitative statements in a declarative language. It is therefore easy to incorporate the complete model in the model-object held in persistent store in the DIS server. The advantage of doing so is that the model is accessible to other applications, such as a process specification system. Being able to see precisely how a model will treat a given step facilitates the model assignment decision. The disadvantage of this approach is the inefficiency of having to fetch the model across the network and reinterpret it each time it is used.

For other simulators, such as SUPREM, the models are embedded in the application in the form of compiled code. For inspection purposes, the model-object could at best contain a redundant description of the mathematical model that is so embedded.

At a minimum, the model-objects should contain three types of information:

1. an indication of the model to be applied, whether by name, or description;
2. indications of how parameters of the step object map into the parameters of the model (this is at least a mapping between parameter names, but might also require an indication of mathematical relationships: for example, change in unit of measure);
3. additional information required by the model that is not expected to be present in the process representation (for example, specification of the time step or gridding technique for a finite element type simulator).

An application driven by a representation of the process would extract that representation by a variant of the following procedure.

The application would "walk" the decomposition tree of the process, examining each object in turn, starting with the highest level module. For each object, the application would request from DIS a list of all parameters that belong in the domain of the application. If there are none and the object is a module, the application would

proceed by asking for the substeps of the module and treating each returned object in turn. If there are no application domain parameters and the object has no further decomposition, then the object represents a step that did not inherit any model information. The application has two choices at this point: either it can decide that such steps are to be ignored (as, for example, cleaning steps would be ignored by a numerical simulator); or that the appropriate model for the step must be determined by some other means, such as a classification procedure.

If the object is a module and does have a model parameter for the application, then the application again has two choices: it can choose to process the object as if it were a leaf in the decomposition hierarchy; or it can decide to continue down the decomposition hierarchy. The existence of a model associated with the module indicates that the module *can* be treated as a step, but does not *require* it to be treated as a step. This choice would be based on criteria specific to the application and/or the invocation of the application. For example, the application could keep track of photolithography modules it has processed. The first time a photolithography module is processed it would be expanded into its constituent substeps, despite having a model at the module level. The next time the same module is encountered, the module would not be expanded, and the module-level model would be invoked. In this way, a run-sheet generator could arrange to expand the details of each photolithography module only once, and refer to these expansions when the same modules are used additional times. As another example, a “wrapper” for the SUPREM simulator could decide, based on preferences indicated by the user invoking the program, whether furnace steps are to be simulated as monolithic steps or whether their equipment substeps should be separately simulated.

As mentioned in the previous subsection, the model information associated with a process object might be ambiguous in two different ways. More than one model parameter for the same application might have been inherited. Also, the model object that is the value of a model parameter might be an interior node in the model specialization hierarchy, representing an abstract model for which there are several specializations to choose from. Different applications might treat these situations differently.

The existence of multiple models might require that the application decide which is most appropriate, or it might require that all models be invoked. For example, a

simulator that could simulate oxidation and diffusion, but not their interaction, might be set up to treat an oxidation-enhanced-diffusion step as two steps performed in either order: an oxidation step and an independent diffusion step. The OED step would have both the oxidation and diffusion models associated with it for this simulator, and both would be invoked. In most cases, however, the application would be designed to select the most appropriate model based on further examination of the characteristics of the step.

Similarly, the abstract model situation may be handled differently by different applications. For some applications, the abstract model can be invoked directly. For others, the application will have to decide which of the specializations of the model is most appropriate based on the characteristics of the step.

In the simplest scenario, only process objects with a model parameter are processed, each such object has or inherits only one model parameter in the domain of the application, and that model has no specializations. Then the existence of a model parameter serves to decide whether modules are expanded and whether steps are processed, and the value of the parameter indicates how the object should be processed. However, as the discussion above indicates, using this mechanism does not preclude employing more sophisticated techniques to solve the model selection problem.

### 5.4.4 Implementation of Model Selection

In this subsection we describe how the model selection task was implemented for the process representation defined by the Specification Editor developed at Stanford by Robert Hartzell.

A simple version of the model selection process described in the previous sections was implemented in a pass-through server that provides read-only language-independent DSL level access to the representation. This server is called the Language Independent Semiconductor Process Representation Server (LISPRS). This server is written in C++ so that it can directly employ a C++ API created to facilitate access to the representation by programs written in C++. Figure 5.8 illustrates the scenario. The LISPRS server accepts commands in a Specification Editor-specific DSL language, and employs the C++ API to translate them into AOI calls on the representation as stored in DIS.

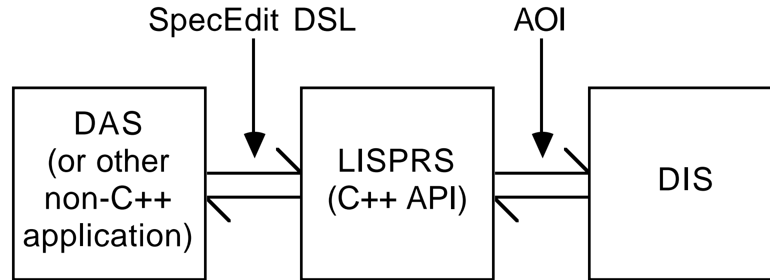


Figure 5.8 Language-Independent Semiconductor Process Representation Server

The principal command provided by the LISPRS read-only pass-through server is called “WALK-MODULE.” This command traverses the composition hierarchy of a module, collecting into a list all steps that have parameters belonging to a specific domain, along with the names and values of those parameters. By this method, a flattened description of the process representation filtered for use by a specific application is fetched in one step, rather than being fetched incrementally. It allows for efficient access of the module structure without the need of complex cache management in the pass-through server.

The Specification Editor makes a high-level distinction between Modules, Steps and Equipment-Substeps. Modules can be composed only of other modules and steps. Steps must be atomic, or composed of Equipment-Substeps. Equipment-Substeps must be atomic. The representation currently only permits model information to be associated with objects that represent Steps. When a step has no model associated with it, either directly or via inheritance, the LISPRS server does not include the step in the flattened description. This implies that the policy regarding such steps is that they are not to be processed, rather than that some other mechanism should be invoked to determine the most appropriate model. Note that one could implement the alternate policy by associating the application’s root model-object with every step object as a default model via inheritance. Then every step would appear in the list, and the root model-object would be a highly ambiguous model assignment to be resolved by other means.

Currently, the DAS simulator associates at most one model-object with each step, and the model-object associated is not abstract. Thus, there are no ambiguous model assignments.

The flattened description returned by the LISPRS WALK-MODULE command is a list, with a sublist for each step to be simulated. The sublist for each step contains the name of the step and the names and values of each parameter marked as being in the DAS domain, including the unique model-parameter with the name of an unambiguous model-object as value.

From this list, a process specification suitable for DAS is constructed. For each step, an expression similar to that in figure 5.2 is generated from information in the sublist for that step and information in the model-object. The name of the model-object serves as the name of the process being simulated, e.g. OXIDATION. The name of the step object serves as the name of the interval.

The model-objects have a slot for each of the parameters that might appear on a step they are associated with that are relevant for the simulation of the model. The value stored in each of these slots is a *parameter-translation* expression indicating how the value and other attributes of the step parameter should be presented to the simulator. The expressions take the form of constraints for the simulator written in the lisp "backquote" syntax. For example, the DAS-ETCH model-object has a slot called "`Chemical Bath`" because some of the etch step objects (in particular the WET-ETCH% objects) with which this model-object can be associated have parameters with this name. The value of the slot is the string:

```
"`(called $etchant ,value)".
```

The backward quote (`) indicates that the expression, when it is read by the lisp reader and evaluated, should be taken "as is," i.e. not evaluated, except that subexpressions within the expression preceeded by a comma should be evaluated and replaced in the expression by their value. This string is fetched during the processing of the description of an etch step object, read by the lisp reader, and evaluated in a context in which the symbol `VALUE` is bound to the value of the step's "`Chemical Bath`" parameter. This produces a constraint that will be passed to the simulator of the form, e.g.:

```
(CALLED $ETCHANT |6:1 Buffered HF|)
```

which states that the `$ETCHANT` parameter of the `DAS-ETCH` model is that `ETCHANT` called `|6:1 Buffered HF|`.

The parameter-translation expression can access attributes of the parameter, as well as its value. For example, the "`Time`" parameters of oxidation and diffusion steps are translated by the string:

```
"`(N= $duration ,(minutes value (attribute units)))".
```

In this expression, the subexpression following the comma is a lisp S-expression that calls the function `MINUTES` (a function that converts times to minutes) with two arguments. The first argument is the value of the symbol `VALUE` (which is bound to the parameter's value). The second argument is the value of the form `(ATTRIBUTE UNITS)`. In the context in which the evaluation takes place, `ATTRIBUTE` is a macro that takes the name of an attribute and returns the value that the parameter has for the named attribute. The resulting constraint passed to the simulator specifies that the model's `$DURATION` parameter is numerically equal to the value of the step's parameter expressed in minutes.

More complicated translations are possible and necessary. For example, some etch steps have a parameter called "`Etch Material`." The import of this parameter is that the etch step is intended to etch this material, whatever other materials it might etch. The simulator is informed of this by the following parameter-translation string:

```
"`(for-all-existing $l \\: layer-region (start-of (process interval))
   (:=> (is-type ,value
                 (@ (material $l) (start-of (process interval))))
        (> (etch-rate $l (process interval)) 0.0)))"
```

This states that if the material type of any layer-region is the value of the step's "`Etch Material`" parameter, then the etch-rate for that layer-region is positive.

There are two possible ways to deal with the issue of additional constraints. One is to associate the constraints with the step object in the process representation, by having a DAS-CONSTRAINT parameter in the DAS domain whose value simply lists the constraints. The other is to create specialized versions of the model-objects and associate the constraints with these specialized model-objects. The former technique puts information into the general process representation that is highly specific to one application, the DAS simulator. It can be argued that the information is potentially of genuine interest to many applications, as it pertains to the goals and expected outcome of performing the step. However, the language used to represent the information is per

force highly DAS-specific. The latter approach would lead to a proliferation of model-objects that differed only in the particulars of context-specific constraints. Many constraints are specific to the context in which a step is used within a process.

My own preference and recommendation is to use both approaches. A specialized model-object should be constructed when the constraints to be represented can be written in a general fashion, so that the model-object can model a class of steps in multiple contexts. Constraints that are highly context-specific should be associated with the invocation of the step within a module. That is, constraints should migrate from the step object to the model-object when they define a natural class of steps, and this would gain efficiency by avoiding repetition.

There is one additional issue that we have not so far discussed. That is the extraction of mask information from the process representation. Each photolithography module contains a mask-expose step, for which the specific mask to use is a parameter that must be specified in the module. Several things complicate this issue. First, each different application may require different information concerning the mask. A CIM electronic runsheet system requires an identification of the glass reticle that is to be used in the fabrication facility. A scheduling system need only be made aware of the number and identity of copies of these reticles available. Simulators require that the geometry of the mask be represented electronically. Furthermore, simulators with different capabilities require different versions of the geometry. A one-dimensional simulator merely needs to know whether the mask is opaque or transparent in the region being simulated. A two-dimensional simulator needs to know the one-dimensional mask information along the cut-line whose cross-section is being simulated. The DAS simulator falls into this category. Finally, a three-dimensional simulator requires the geometry of the mask in two-dimensions for the structure being simulated.

Second, semiconductor processes are often employed to manufacture more than one product. Also, there will in general be more than one structure, cut-line or region for simulators to simulate. For each product there will be a distinct set of reticles to be used. For each structure, cut-line or region to be simulated, there will be a distinct set of mask descriptions.

Finally, the masks define circuits and structures. Although these must be designed with a specific class of manufacturing process in mind, the development of

processes and masks are independent from the perspective of version tracking and configuration management.

For these reasons, the “mask” parameter in the mask-expose steps cannot specify the mask information directly. Instead, the parameter must identify this information indirectly, by specifying a “mask layer.” The SPR server must represent masks and mask-sets as independent design objects, and allow for different classes of these objects that specify the information needed by different classes of applications.

These issues are discussed at greater length in [9] . That paper includes a suggestion regarding the protocol to be used for creating and accessing mask information. The ideas presented in that paper were adopted to a large degree by the Specification Editor. However, the Distributed Information Server (DIS) has as yet made no commitment regarding the representation of mask information. In particular, the Domains Specific Language (DSL) for the semiconductor domain was not augmented to implement a version of the suggested protocol.

For the purposes of this work, we developed lisp functions that invoke the Abstract Object Interface protocol to implement our suggested DSL protocol for mask information on the client (application) side of the client-server boundary.

# Chapter 6. Architecture for Integration

In previous chapters the strengths and weaknesses of representations of both experiential and theoretical knowledge have been discussed. In this chapter we make the argument that these strengths and weaknesses are complementary for the two types of knowledge. To emphasize the complementary character of the two types of knowledge, we introduce the concepts of Phenomena Scope and Situation Scope as two dimensions along which knowledge representations can be classified.

We then present a key contribution of this work, which is one particular approach to combining experiential and theoretical knowledge that we call Generic Rules. Generic rules enable a synergistic interaction between experiential knowledge and theoretical knowledge, capturing the strengths of both.

## 6.1. Phenomena Scope and Situation Scope

Let us briefly review the strengths and limitations of both forms of knowledge. For experiential knowledge, represented in heuristic rules or causal associations, a major strength is flexibility. Any phenomenon of interest can be included, provided only that it can be named, and that it can be related to other phenomena, at least in some particular context. It is not necessary to capture all ways that it might relate to other phenomena: partial descriptions can be of use.

The primary limitation of experiential knowledge is brittleness, defined as the inability to adapt knowledge encoded for one context to another context. The representation of a causal relationship between two phenomena often does not represent how that relationship is mitigated by other factors. When a change in context entails a change in such other factors, the represented relationship may be invalidated. However, there is no way to tell. Furthermore, the lack of a general mechanism for expressing how classes of concepts are interrelated leads to the use of overly-specific terms. This unduly restricts the applicability of relevant knowledge in a new context: the relevance of the knowledge to the new context might not be recognized. In both cases, alleviating the

problem requires representing knowledge of the domain with greater detail, at a finer level of granularity.

The chief advantage of theoretical knowledge is a reduction in brittleness. More of the factors that influence the relationships among phenomena are explicitly represented. Moreover, the effect of changes in these factors can often be predicted. Thus, representations of theoretical knowledge can be applied to a wider range of new contexts with a greater degree of confidence.

As we concluded in Chapter 4, the main limitation of our representations of theoretical knowledge is that they do not treat all phenomena. It is a feature of the models used in model-based techniques that they only represent a subset of the domain: the subset is chosen so as to concentrate reasoning on those features and phenomena in the domain that are relevant to the problem-solving task. The trade-off is to accept a restriction in the scope of the problems that can be solved in order to achieve tractability and efficiency in solving them. This restriction in scope is not only desirable, but necessary: we can only model some phenomena properly if we adopt a very fine-grained ontology that has a severe impact on computational complexity; we have not yet devised satisfactory ways to model some other phenomena; and we do not understand still other phenomena sufficiently to model them at all. Our understanding of phenomena and our understanding of how best to represent and reason about phenomena will both increase over time. However, it will always be case that some phenomena lie outside the reach of our either our theoretical understanding or our representation and reasoning ability.

For illustrative purposes, we define two terms: ***phenomena scope*** and ***situation scope***. The term phenomena scope is meant to be a measure of the flexibility of a knowledge representation paradigm. It is a count of the number of phenomena that can be usefully captured within the paradigm. Rule-based systems that encode experiential knowledge tend to have high phenomena scope: many different phenomena can be included in the representation, because very little detail concerning each phenomenon needs to be encoded. Representations of theoretical knowledge, on the other hand, attempt to encode a larger amount of detail concerning each phenomenon, sufficient to constitute a predictive model of how the phenomenon relates to other phenomena in many different contexts. The difficulties involved in creating such representations often

limit the range of phenomena that can be captured. Systems based on representations of theoretical knowledge thus tend to have low phenomena scope.

The term situation scope is meant to be a measure of the robustness of a knowledge representation paradigm. It is a count of the number of distinct situations or scenarios — what I have been referring to as contexts — for which the knowledge can be considered to be valid. As we have noted above, systems based on representations of experiential knowledge tend to be brittle. The physical mechanisms underlying the heuristic associations in such systems are often not represented with sufficient detail to enable the system even to determine whether the associations are valid in a new context, much less how their conclusions need to be modified for the new context. In contrast, theoretical knowledge often takes the form of predictive models that explicitly represent the physical mechanisms underlying behaviour. Such systems therefore tend to have a relatively high situation scope.

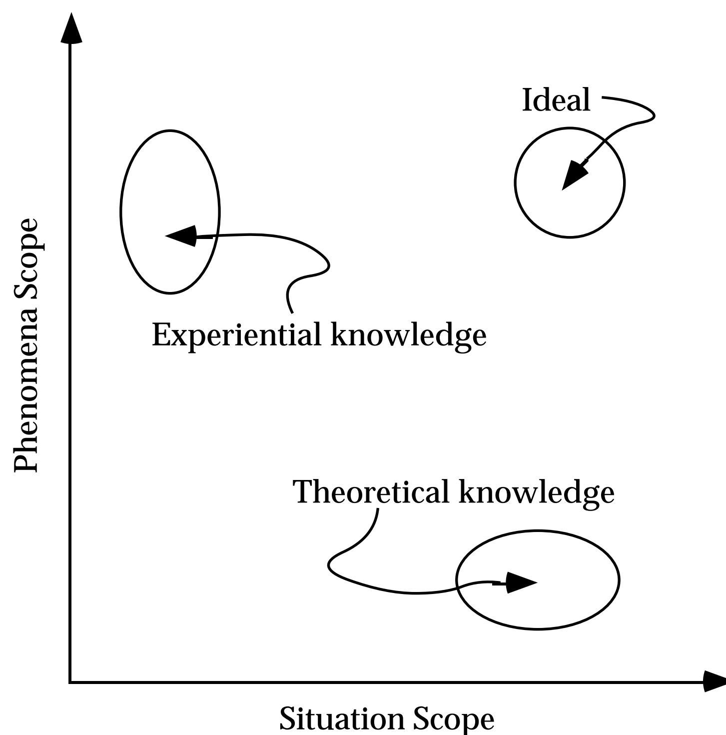


Figure 6.1 Graph of Phenomena Scope vs. Situation Scope

Figure 6.1 depicts phenomena scope and situation scope as two orthogonal axes defining a two-dimensional space of potential reasoning systems. As shown in the figure, systems based on experiential knowledge cluster in the upper left portion of the

quadrant, having high phenomena scope but low situation scope. Systems based on theoretical knowledge cluster near the lower right portion of the graph, having high situation scope but low phenomena scope. The ideal system would fall in the upper right hand portion of the graph: capable of representing many phenomena of interest as they occur and behave in a wide variety of situations.

We have defined these two ways of characterizing knowledge-based systems in order to highlight the fact that our two fuzzy categories of knowledge, experiential and theoretical, and their respective representations, high-level heuristic rules and models, have complementary attributes with respect to these characteristics. The weakness of one is the strength of the other, and vice versa. It is therefore natural to contemplate methods for incorporating both into a single system, in a manner that would allow the strengths of each to overcome the weaknesses of the other.

## 6.2. Generic Rules

A key contribution of this work is the identification of one such method for the synergistic integration of these two forms of knowledge. The instrument of the method is a representation we call the *Generic Rule*.

The intent of a generic rule is to represent a piece of knowledge concerning a phenomenon that lies outside the scope of the available representations of theoretical knowledge in a manner that preserves some of the robustness conferred by such representations. The generic rule accomplishes this by combining the features of both experiential and theoretical knowledge representations.

The generic rule, like the rules normally employed to represent experiential knowledge, is a high-level heuristic association between characteristics of a situation and conclusions that might be drawn concerning that situation. It differs from an ordinary rule in that it employs the vocabulary and reasoning machinery of a representation of theoretical knowledge in order to explicate, to the degree that the theoretical knowledge is capable of doing so, how the inference sanctioned by the rule depends on contextual information. A generic rule represents a general pattern for a class of heuristic rules that can be instantiated for a given context.

Let us illustrate with an example. In Chapter 3 we introduced the example of a causal association found within the AESOP PIES knowledge base. We repeat the association here:

```
POLY-GATE-UNDER-ETCH → CHANNEL-LENGTH-LONG.
```

- **Experiential representation is not robust:**

In our discussion of this example we noted that this association is not robust, because its validity depends on a series of relationships holding among the steps of the manufacturing process and the structures on the wafer, which are not explicitly represented in the association. We also noted that the terms employed in the association were overly specific, to compensate for the fact that the PIES representational machinery offers no mechanism for determining in general which etch operations are causally involved in the shaping of which structures on the wafer.

- **Theoretical representation cannot capture phenomena involved:**

Our representation of theoretical knowledge described in Chapter 4 provides a mechanism for representing some of the information that is missing: which etch operations affect which wafer structures. Further, it provides a mechanism for determining this information from a representation of the manufacturing process. However, the phenomenon at work in this causal association lies outside the scope of the phenomena that can be captured in our models. The phenomenon involves lateral etching and sloping sidewalls, an inherently 2-dimensional geometric feature. Our models do not admit any lateral effects of processing, and our representation of wafer structure, though topologically 2-dimensional, is geometrically only 1-dimensional.

Thus, our causal association representation of experiential knowledge can represent the phenomenon, but not in a robust fashion, and our model-based representation of theoretical knowledge cannot capture the phenomenon.

- **Phenomena can be captured robustly in Generic Rules:**

We can eliminate the overly specific terms in the causal association by introducing variables. Denoting variable names with a '$' prefix, the new, more general rule would read:

```
($etch-step UNDER-ETCH)→($wafer-structure LONG).
```

Treating the variables as universally quantified, the rule can now be applied to any process by instantiating the variables with etch steps and wafer-structures associated with that process. Note that the process representation lists all etch steps, and the representation of wafer structure computed by the Discrete Action System provides many possible candidates for the overly-long wafer structure. The DAS system also provides the machinery necessary for finding instances of such general statements.

The difficulty with this approach is that the general rule will have many invalid instantiations. For example, one instantiation might posit a causal relationship between under-etching during the 'via' etch step and excessive length of the bipolar transistor's collector region, whereas such a relation is highly improbable, if not impossible. However, we can attempt to eliminate invalid instantiations by specifying constraints that must be satisfied by the objects substituted for the variables. Here it is important to note that, while it is not possible to capture the phenomena addressed by the original rule in the DAS representations, it is possible to use the DAS representations to describe most of the conditions that must be satisfied in order for those phenomena to occur.

**DEFINITION:** A *Generic Rule* is general pattern for a heuristic rule, employing variables within the normal syntax for heuristic rules (in our case, causal associations), together with a list of constraints that must be satisfied by potential values for the variables. The vocabulary, syntax and reasoning machinery employed for theoretical reasoning is used to specify the constraints and to determine whether they are satisfied within a given context.

Figure 6.2 illustrates the syntax for specifying a generic rule. A Generic Rule is itself a rule of the form "IF <conditions are met for bindings of variables> THEN <posit rule-patterns with variable bindings substituted>." However, the subject matter of the rule is the formation of another rule. Thus, a Generic Rule might be considered a "meta-

rule." However, the term meta-rule is reserved for rules concerned with control strategy during reasoning. Since a generic rule describes an entire class of potential rules, we prefer the adjective "generic," in its sense: "descriptive of an entire group or class." [5]

```
(defGenericRule <rule-name>
  :variables ((<var> <type>) (<var> <type>) ...)
  :conditions (<condition 1> <condition 2>  ...)
  :rule-patterns (...))
```

Figure 6.2 Syntax for Generic Rule

The conditions necessary for the phenomena involved in our example rule are listed in Chapter 3. Let us examine them each in turn.

The first condition concerns the degree of anisotropy of the etch step. The example original rule concerns the relationship between the length of an etch step and the "horizontal" length of a physical structure. Thus, a necessary condition for the phenomenon to occur is that the etch step in question involve lateral etching. The DAS models cannot determine when an etch step will involve lateral etching. However, it is reasonable to require that the process specification (the manufacturing plan) include this information, either directly or indirectly. In an object-oriented process representation system, the degree of anisotropy expected can be one of the traits by which etch steps are classified, so that this trait can be specified for abstract etch steps and inherited to specific etch steps. The Specification Editor process representation currently includes a parameter call "ETCH-ANGLE" that is intended to capture the degree of anisotropy by specifying the expected angle of the sidewalls. The test for the first condition can be expressed as an ordinal relation between this etch-angle parameter and a suitable threshold (beyond which the lateral etching is considered insignificant). Thus, the first condition can be written:

```
;; Condition 1:
(:AND (is-type (process $etch-step) etch)
      (< (ETCH-ANGLE $etch-step) ANISOTROPY-THRESHOLD)).
```

---

[5] American Heritage Dictionary — Standard Edition

The second condition is that the etch operation is masked. This condition is determined by the state of the wafer when the etch step is executed: are some parts of the wafer covered with a material that doesn't etch well and other parts not? The state of the wafer at given points in the process, and the impact of each etch step on the materials at the surface of the wafer is determined in the course of qualitatively simulating the manufacturing plan using the DAS models. Thus, this condition can be expressed and tested within DAS. There are many ways this condition might be expressed. Since we wish to identify the layer-region that is laterally etched, the condition might best be expressed from the perspective of this layer-region. The condition can be simply expressed as the conjunction of two conditions: that the layer-region survives the etch step, and that the layer it belongs to is changed by the etch step (i.e. some layer-regions in the layer do not survive the etch step):

```
;; Condition 2 (first version):
(:AND
  (:NOT (destroyed $wafer-structure (end-of $etch-step)))
  (member (@ (layer $wafer-structure) (end-of $etch-step))
          (changed-layers $etch-step)))
```

However, we can be more specific than this. If the layer-region is to be etched on both sides, it must be a layer-region that survives the etch step, for which the layer-regions in the same layer in adjacent regions are destroyed:

```
;; Condition 2:
(:AND
 (:NOT (destroyed $wafer-structure (end-of $etch-step)))
 (exists $wr : wafer-region
  (:AND
   (V= (@ (region-type $wr) (end-of $etch-step))
       (@ (region-type $wafer-structure) (end-of $etch-step)))
   (exists $lr e (@ (layer $wafer-structure)
                    (start-of $etch-step))
    (:AND
     (V= (@ (region-type $lr) (start-of $etch-step))
         (@ (region-type (left $wr)) (start-of $etch-step)))
     (destroyed $lr (end-of $etch-step))))
   (exists $lr e (@ (layer $wafer-structure)
                    (start-of $etch-step))
    (:AND
     (V= (@ (region-type $lr) (start-of $etch-step))
         (@ (region-type (right $wr)) (start-of $etch-step)))
     (destroyed $lr (end-of $etch-step)))))))
```

The third condition, that the duration of the step is normally longer than that required to etch through the material in order to effect lateral etching, is easily expressed within DAS. Furthermore, that the condition is satisfied for a given etch step is likely to be specified directly as a constraint in the process specification. Once again, we express the condition relative to the layer-region that is being laterally etched:

```
;; Condition 3:
(exists $lr e (@ (layer $wafer-structure) (start-of $etch-step))
  (> (duration $etch-step) (etch-destroy-time $lr $etch-step)))
```

At this point we should note that the original rule combined the results of two distinct phenomena: lateral etching and masking an implant with the structure that was etched. It would have been better expressed, even in PIES, as two rules: an interlevel association between under-etching during the POLY-GATE-ETCH step (Process level) and increased length of the POLY-GATE structure (Physical-Structure level) and an intralevel association at the Physical-Structure level between the increased length of the POLY-GATE structure and the resulting increase in CHANNEL-LENGTH. The two rules would be expressed in PIES as follows:

```
POLY-GATE-UNDER-ETCH → POLY-GATE-LONG
POLY-GATE-LONG → CHANNEL-LENGTH-LONG.
```

A generic rule for the interlevel association would take the form indicated in figure 6.3.

```
(defGenericRule Insufficient-Lateral-Etch
  :variables
  (($etch-step dynamic-interval)
   ($wafer-structure layer-region))
  :conditions
  (<Condition 1> <Condition 2> <Condition 3>)
  :rule-patterns
  ((($etch-step UNDER-ETCH) -->
    ((width $wafer-structure) high))))
```

Figure 6.3 Interlevel Generic Rule for POLY-GATE-UNDER-ETCH example

The fourth condition, that the process is self-aligned, applies only to the second, intralevel rule. The condition must indicate that in some areas the implantation is masked by a structure on the wafer other than photoresist. Figure 6.4 illustrates one way to encode this condition and the generic rule for the intralevel association. Although the causal association requires only two variables, for the two structures involved, it is

necessary to include a third variable identifying the implantation step by which the two structures are interrelated. The two structures are correctly identified if the masking structure is at the surface of the wafer in the same regions as the masked structure and is not photoresist, both structures are in layers affected by the same implantation operation, and the masked structure is not itself modified by the operation.

```
(defGenericRule Structure-Masked-Implant
  :variables
  (($implant-step dynamic-interval)
   ($masking-structure layer-region)
   ($masked-structure layer-region))
  :conditions
  ( ;; The step is an implantation operation
   (is-type ion-implantation (process $implant-step))
   ;; The masking structure is at the wafer surface in the same region-type
   ;; as the masked structure.
   (V= (@ (surface (region-type $masked-structure))
          (start-of $implant-step))
       $masking-structure)
   ;; The masking structure is not photoresist
   (:NOT (is-type photoresist
                  (@ (material $masking-structure)
                     (start-of $implant-step))))
   ;; The layers that the masked and masking structures belong to are both
   ;; affected by the operation
   (member (@ (layer $masked-structure) (start-of $implant-step))
           (new-layers $implant-step))
   (member (@ (layer $masking-structure)(start-of $implant-step))
           (new-layers $implant-step))
  ;; The masked structure is not implanted.
  (> (ion-bottom (@ (region-type $masked-structure)
                    (start-of $implant-step))
                 (start-of $implant-step))
     (@ (top-pos $masked-structure) (start-of $implant-step))))
  :rule-patterns
  ((((width $masking-structure) HIGH) -->
     ((width $masked-structure) HIGH)))))
```

Figure 6.4 Intralevel Generic Rule for POLY-GATE-UNDER-ETCH example

Note that both rules refer to the widths of structures being too “high.” We could have written the rules in terms of the lengths of structures being too long. However, in the general case we do not know *a priori* whether the dimension in question is a width or a length. Within the ontology of DAS, all lateral dimensions are widths. Whether the lateral dimension of a particular structure is considered a width or a length depends on the conventions for the device of which the structure is a part, and on what the orientation is of the simulated cut-line relative to the standard orientation of the device. In the case of the POLY-GATE example, the lateral dimension in question is considered

a length because this is the dimension along which current will flow through the device. If the cut-line cuts the device longitudinally—along the path between source and drain—then the width of the layer-region in DAS representing the gate corresponds to the “length” of the gate. Similarly, the silicon layer-region below the gate is the channel, and if the cut is longitudinal then the (DAS) width of this layer-region corresponds to the CHANNEL-LENGTH. To recover the rules precisely, a *concept classifier* is needed. The concept classifier must have sufficient knowledge to recognize when a structure, as it is described in DAS, corresponds to a named device feature. The concept classifier is described in the section on system architecture below.

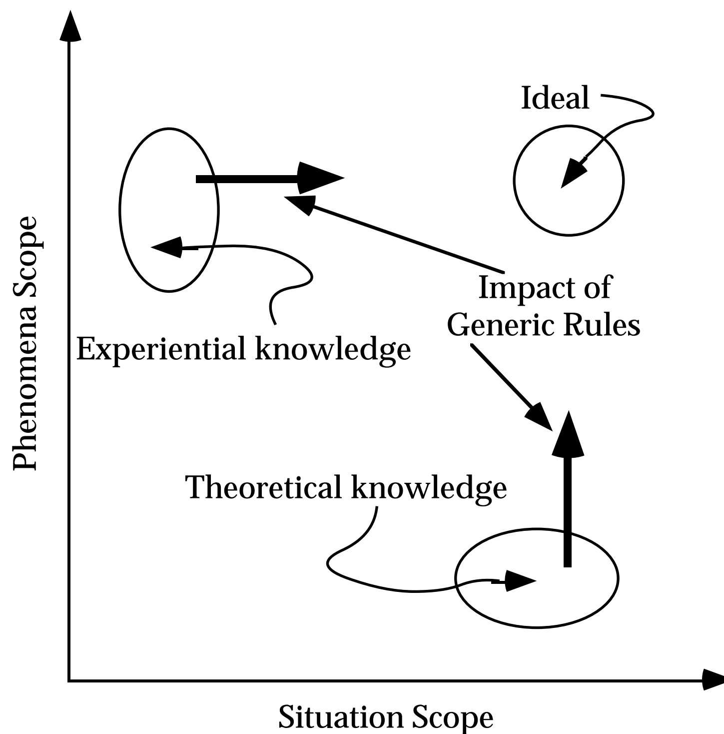


Figure 6.5 Impact of Generic Rules in Phenomena/Situation scope space

### 6.3. Impact of Generic Rules

We can view the impact of generic rules in several ways.

From the perspective of systems based on experiential knowledge, generic rules provide a mechanism for using model-based techniques to increase the robustness of heuristic rules, by representing what is known theoretically about when and where the phenomena of interest in each rule can be expected to occur in new situations. From this perspective, generic rules have the impact of increasing the situation scope of

experience-based systems. This is represented by the arrow pointing to the right in figure 6.5.

From the perspective of systems based on theoretical knowledge, generic rules provide a mechanism for encoding partial knowledge about phenomena that would otherwise lie outside the scope of the represented theory. From this perspective, generic rules have the impact of increasing the phenomena scope of theory-based systems. This is represented by the arrow pointing upward in figure 6.5.

In each case, the addition of generic rules moves the system closer to the ideal.

A third way to view Generic Rules is to note that they form a mechanism by which experiential and theoretical knowledge can be combined in a synergistic fashion to accomplish what neither alone can accomplish: robust codification of heuristic and theoretical knowledge about a wide range of phenomena.

## 6.4. System Architecture

Figure 6.6 illustrates, as a data-flow diagram, the architecture that supports integration of theoretical knowledge and experiential knowledge using generic rules. The rounded-corner rectangles represent collections of data of varying kinds. The square-corner rectangles represent computations. Arrows from data rectangles to computation rectangles represent that the data are inputs to the corresponding computations. Arrows from computation rectangles to data rectangles represent that the data are outputs of the indicated computations.

The Language Independent Semiconductor Process Representation Server (LISPRS), described in Chapter 5, extracts a representation of the manufacturing plan from the Distributed Information Server (DIS). This representation of the plan, including mask information, and the process models serve as inputs to the Discrete Action System (DAS), which assimilates the plan by qualitatively simulating it. The output of this computation is represented in the diagram as a "Dependency Graph," because its main purpose is to serve as a repository of information regarding the causal dependencies in the plan. However, it also includes complete historical information regarding what the wafer structure looked like at various points in the process (specifically at each processing step interval boundary), when changes were made to the wafer structure and

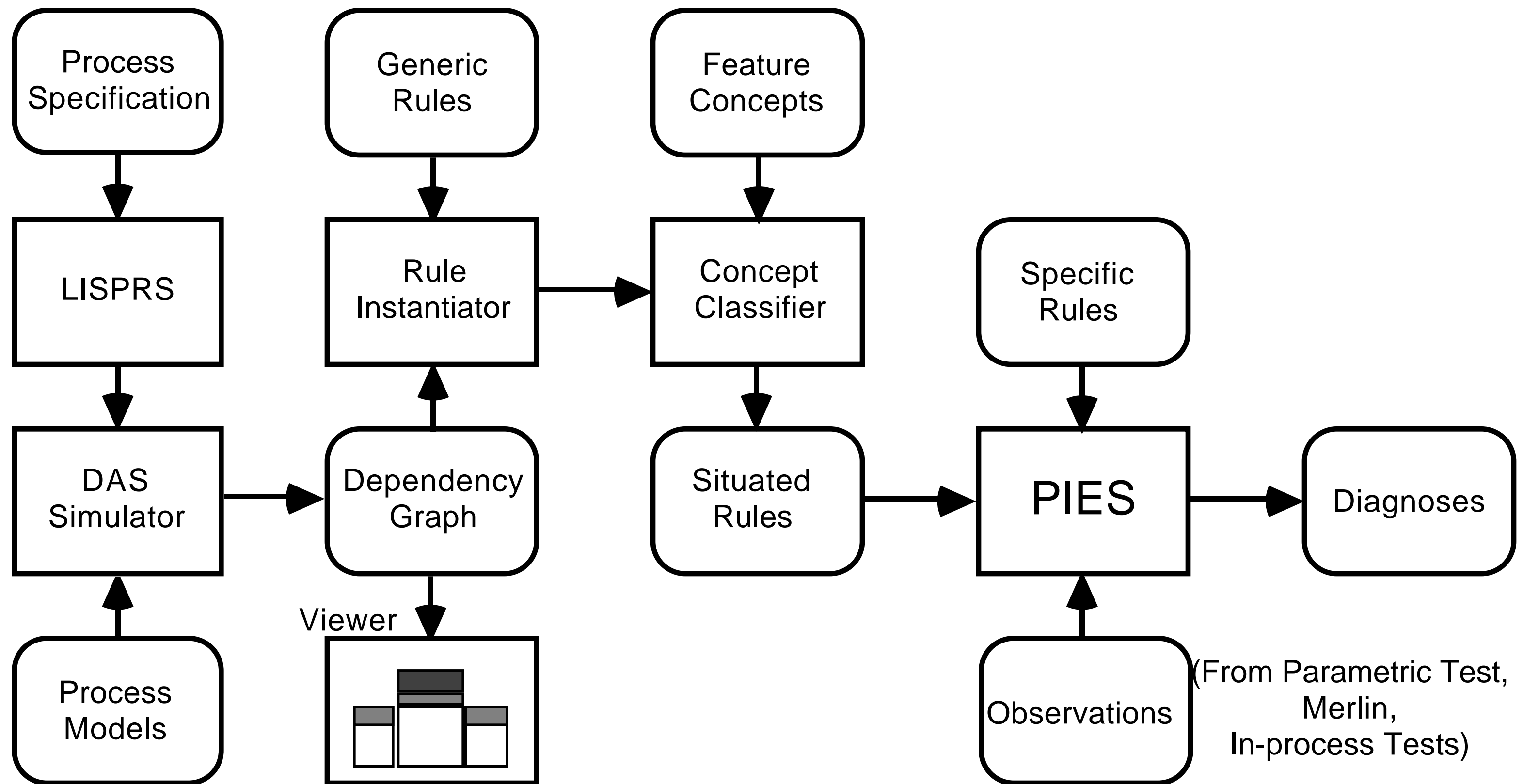


Figure 6.6: Architecture for Integration Using Generic Rules

why, and what the sequence of processing was. Thus, a Viewer can display the wafer structure at the end of each step and respond to queries, as described in Chapter 4.

The Dependency Graph and a knowledge base of Generic Rules form the inputs to the Rule Instantiator. The Rule Instantiator finds acceptable instantiations of each of the generic rules and feeds them to the Concept Classifier. As shown on the diagram, the output of the Concept Classifier is a set of Situated Rules. The Parametric Interpretation Expert System (PIES) can then employ both the Situated Rules and Specific Rules to produce Diagnoses from Observations.

Situated Rules are the instantiations of Generic Rules to the situation represented by the process specification. Specific Rules are the normal context-specific PIES causal associations. Although the diagram shows Situated Rules and Specific Rules as two distinct sets of data, in fact the principle role of the Concept Classifier is to merge the Situated Rules into the PIES knowledge base.

A PIES causal association network is most effective when the network has a high degree of connectivity. That is, when there are many associations relating the various fault concepts. The greater the number of causal links that a fault concept has to manifestations, for example, the more opportunities the system has to seek verification of the concept as a diagnostic hypothesis. It is therefore important that when two fault concepts are equivalent that they are identified and represented uniquely. This is also important to avoid having two equivalent fault concepts competing for support as the correct diagnosis.

In the PIES system itself, identification of equivalent fault concepts is the responsibility of the knowledge enterer. It is aided by the capability to classify fault concepts within a taxonomic structure. The role of the Concept Classifier is to fix the position within the taxonomy of the fault concepts mentioned within the situated generic rules, and to recognize when a rule refers to a pre-existing fault concept.

In support of this function, a secondary role of the concept classifier is to recognize when fault concepts (or the structures or processing steps they involve) in the rules correspond to concepts for which process engineers have preferred names, and to substitute these preferred names. For this reason the Concept Classifier has a second set of inputs, entitled “Feature Concepts” in the illustration. This is a glossary of preferred

names for features, along with information regarding how to recognize them in DAS descriptions.

# Chapter 7. Experimental Results

This chapter describes the experiments performed to test the main theses of this dissertation empirically. The first section describes two experiments. The results of performing these experiments are documented in the second section. A third section presents a summary of the results.

## 7.1. The Experiments

The work reported in this dissertation is about the robust representation of experiential knowledge and theoretical knowledge concerning diagnosis of stable semiconductor manufacturing processes. There are therefore two key hypotheses to test empirically:

1. That the knowledge is represented in a robust manner. That is to say, that the theoretical knowledge can be applied to a new manufacturing process, and that experiential knowledge gained from experience with one manufacturing process can also be applied to a new manufacturing process; and
2. That the knowledge is effective for the task of diagnosing a stable semiconductor manufacturing process.

The empirical portion of this work thus consists of two experiments. The first experiment demonstrates that a diagnostic knowledge base can be adapted to a new semiconductor process. The second demonstrates that this adapted knowledge base can effectively diagnose manufacturing problems with the new process.

### 7.1.1. Adapting Diagnostic Knowledge to a New Process

To test the first hypothesis, we must apply the knowledge we have represented to a new semiconductor manufacturing process. For the theoretical knowledge, the hypothesis is considered verified if the models can be used to assimilate the process and build a dependency network of the causal relationships inherent in the process. For the experiential knowledge, it is necessary to have a body of experiential knowledge relevant to another process. The experiment consists of using the ontology of our theoretical models to represent the rules in the experiential knowledge base for the other process with our Generic Rule representation technique. Then these generic rules are

instantiated in the context of the new process. The hypothesis is considered verified if the instantiation process produces rules that are acceptable for the new process, and produces few or no inappropriate rules.

The acceptability of the causal relationships produced by the assimilation of the process, and the causal associations produced by the instantiation of generic rules is determined in two ways. First, the subjective opinion of an expert is consulted. Second, the diagnostic knowledge so obtained is tested by applying it to the diagnosis of manufacturing faults in the second experiment described in the next subsection.

For the purpose of these experiments, we take the Stanford BiCMOS process [50] as our representative of the new process. We employ the AESOP PIES knowledge base [11] as our source of experiential knowledge regarding another process. The AESOP knowledge base was created by performing a series of simulation experiments for the Stanford CMOS process. The BiCMOS process was developed as a modification of the CMOS process. It is therefore reasonable to assume that much of the knowledge gained regarding the CMOS process should transfer to the BiCMOS process.

However, the BiCMOS is not a trivial incremental variation of the CMOS process. The CMOS process creates two types of active devices on the wafer: p- and n-type field effect transistors. The term CMOS is an acronym that means "Complementary MOSFET," where MOSFET is itself an acronym for "Metal Oxide Semiconductor Field Effect Transistor." The adjective "complementary" in the name refers to the fact that the p- and n-type transistors have complementary characteristics: the same voltage turns one on and the other off. The BiCMOS process creates these two types of transistor as well as a third: an NPN bipolar transistor. The "Bi" in BiCMOS refers to this bipolar device. To create this third type of device, the BiCMOS process augments the CMOS process with additional steps. Also, to minimize the number of additional photolithography operations, the steps in the CMOS process were modified and rearranged to some degree so that steps that create parts of the CMOS devices simultaneously create parts of the NPN bipolar device. For this reason, some causal associations in the AESOP knowledge base may not apply directly to the BiCMOS process. Also, the phenomena at work in some of the causal associations may actually also occur in the manufacture of the bipolar structures.

#### 7.1.1.1 Assimilating the Stanford BiCMOS process

Our source for the semiconductor process representation of the BiCMOS process is the Specification Editor database managed by the Distributed Information Service. Our methodology for extracting a representation suitable for the Discrete Action Simulator involves modification of the database. In particular, it requires that:

1. we add the "DAS" domain to all parameters that are of interest to the DAS simulator (this is how the LISPRS server identifies which parameters to report);
2. we add two new parameters to the root object of the step library, called DAS-MODEL and DAS-CONSTRAINTS;
3. we specify values for the DAS-MODEL parameter in suitably abstract STEP objects; the value of the DAS-MODEL parameter is the name of a model-object in the database that corresponds to one of the symbolic models; the steps where the model values are stored are chosen so that the correct models are inherited by those steps that we wish to simulate; and
4. we specify values for the DAS-CONSTRAINTS parameter for those steps that require context-specific constraints, as "parameter overrides" in the modules that contain the steps.

Since the database is in daily use by fabrication personnel, these modifications were not attempted on the public version of the database. Instead, a copy of the database was created and kept for the exclusive use of this work. It was hoped that after the modifications were made and tested on the private copy of the database, the same modifications could be made on the public version. To this end, the software that was used to make the modifications was designed to automatically build an executable record of exactly what modifications are made, as well as an executable record of the transactions that would be necessary to restore the database to its original state.

It turns out that the Specification Editor is not compatible with some of these changes. Specifically, the Specification Editor implicitly assumes that parameters may belong to only one domain. It ceases to recognize that a parameter belongs to the SUPREM domain, for example, if the same parameter also belongs to the DAS domain. Thus the precaution of attempting the modifications only on a private copy of the database was warranted. The modifications cannot be transferred to the public database until the Specification Editor is modified.

The representation employed by the Specification Editor does not incorporate the flow control primitives developed by Don Basile [51]. Thus, operations performed only on test wafers are represented by additional steps that, unfortunately, cannot be

distinguished from the steps that operate on product wafers using the Specification Editor API. Since these steps could not be filtered out electronically, it was necessary to make a copy of the process representation from which such steps were deleted. The process also includes other steps that are superfluous from the perspective of the simulator that can be distinguished electronically from steps that are relevant. These include some steps that are performed only on test wafers in order to determine optimal equipment settings. These also include steps that are performed on all wafers, but which have effects that the simulator cannot meaningfully represent, such as cleaning steps, photoresist baking steps, measurement steps and inspection steps. These steps are electronically filtered out of the semiconductor process representation by virtue of not having a non-null value for the DAS-MODEL parameter.

Mask data were created and stored in the database for 2-dimensional slices through three devices: an NPN bipolar transistor, an NMOS transistor and a PMOS transistor. Since the qualitative models cannot treat the kinds of problems that most affect metal layers, only the first 14 (of 20) masks were created and only the first 18 (of 27) modules of the process were fetched. This represents the process up to the creation of the transistor and resistor structures, but before the deposition of any interconnect layers.

This editing and filtering reduces the number of steps in the BiCMOS process from 754 to 96. This indicates the complexity of the process: in our early work we were able to represent a Fairchild bipolar process with only 48 steps. The BiCMOS process requires twice as many steps, and requires more sophisticated models for some of the steps.

These 96 steps were qualitatively simulated on an IRIS workstation with 256 megabytes of random access memory. The simulation took one hour and thirty-one minutes, and the resulting dependency graph takes up 109 megabytes of memory.

#### 7.1.1.2 Generalizing and Instantiating AESOP Causal Associations

The purpose of this part of the experiment is to verify that experiential knowledge gained concerning one manufacturing process can be represented in a manner that enables the transfer of this expertise to a new manufacturing process. The experiment consists of the following steps:

1. we identify a knowledge base containing experiential diagnostic knowledge regarding a particular manufacturing process; in this case, we employ the AESOP knowledge base regarding diagnosis of manufacturing faults in the Stanford CMOS process;
2. for each causal association in the knowledge base, we attempt to discern the criteria that determine the circumstances under which the phenomenon at work in the association will operate;
3. we attempt to encode these contextual criteria using the ontology of our theoretical models as the conditions for instantiation of one or more Generic Rules; more than one Generic Rule may be required if there are identifiable intermediate states that should be represented;
4. we instantiate these Generic Rules for a new process using the dependency network created by the assimilation procedure mentioned in the previous subsection; in our case, the new process is the Stanford BiCMOS process;
5. we examine the causal associations that result from this instantiation process, assessing their quality with the help of an expert.

The focus of this work is on the causal relationships between processing events and the physical structures that they create or modify. We therefore limit ourselves to interlevel causal associations between the process level and the physical-structure level, and to intralevel associations at each of these levels.

In fact, the AESOP knowledge base does not have any intralevel causal associations at either of these two levels. As we discuss in Section 8.2 of the related work chapter, and as we mentioned in Section 6.2 of the chapter on generic rules, the reason for this is the manner in which the knowledge was acquired. A set of experiments with numerical simulators was performed. Response Surface Map methods were employed to codify the results of these experiments as a set of polynomials that represents the multivariate transfer function between the identified inputs and outputs of the manufacturing process. Then causal associations were extracted from this transfer function. This approach treats the entire manufacturing process as "black box," and concentrates on determining the causal relationships between the inputs and the outputs of the box. Thus, this methodology does not discover or represent the intermediate states in the manufacturing process, and no intralevel causal associations are identified.

The knowledge base contains 52 interlevel causal associations between the process level and the physical structure level. These are listed in abbreviated form in Appendix D. Processing problems for only six steps are included: Initial wafer Selection (5 problems), Gate Oxidation (9 problems), N-Channel Threshold Adjust Implant (5

problems), P-Channel Threshold Adjust Implant (4 problems), Polysilicon Etch (4 problems) and N-Well Implant (7 problems) for a total of 34 processing anomalies.

These are related to problems with 6 structures: Gate Oxide (6 problems), P-Channel doping (4 problems), N-Channel doping (4 problems), Channel-Length (4 problems), Substrate doping (5 problems) and N-Well (5 problems) for a total of 28 physical structure anomalies.

The associations fall into three categories. The associations in the first category, containing the bulk of the associations, treat phenomena that are already captured by the theoretical knowledge. Those in the second category treat phenomena that involve topological changes to the process and/or the physical structure of the devices. The models are capable of handling these phenomena. However, the causal pathways generated by the models cannot be trusted when topological changes are involved unless these changes are simulated. The 12 associations in the final category treat phenomena that lie outside the scope of the theoretical models.

#### 7.1.1.3. Category 1: Associations within scope of theory

The AESOP knowledge base contains 34 causal associations in this category. An example of a causal association in this category is the following:

```
Gate-Ox-Temp-High → Gate-Oxide-Thick (Maybe)
```

This association reflects the fact that the thickness of the gate oxide is determined by (among other factors) the temperature at which the oxidation is performed. The theoretical models “know” that there is a positive proportionality between the temperature of any oxidation step and the thickness of the oxide layer that results from that step. Thus, this association clearly falls within the Phenomena Scope of the models. The contextual information required to identify instances of this phenomenon is the following:

1. the oxide layer in question is created by a high-temperature oxidation process (rather than low temperature deposition); and
2. the step in question is the one during which the layer was created.

The causal associations do not explicitly mention time. In this particular association, the oxide layer survives to the end of the process (it is not etched away) and

no additional oxidation affecting this layer takes place. To reproduce this rule exactly we might wish to add such conditions. However, it may be useful to create a more general rule regarding sacrificial oxide layers and oxide layers that see several cycles of growth. Whether instances of such a rule are interesting depends on whether the thickness of such oxides ultimately affects the characteristics of the devices created or the variability of other processing steps.

#### 7.1.1.4. Category 2: Associations involving topological change

The AESOP knowledge base contains six causal associations in the second category. The following is an example of an association in this category:

`N-Vt-II-Not-Masked` → `P-Ch-Conc-High (Maybe)`

This association states that if the PMOS active areas are not protected by photoresist (because the photolithography step was omitted or not masked) when the N-Channel threshold adjust implant is performed, then the concentration of dopant in the channel of the PMOS transistor might be too high. The assimilation of the manufacturing plan would indicate that in the PMOS active regions, the N-channel threshold adjust implant does not reach the silicon layer because it does not get through the photoresist layer. Further, there would be a logical dependence indicating that the presence of the photoresist is predicated on the performance of the DAS-SPIN-ON-RESIST (photoresist coat) operation, and the fact that the region was not exposed during the intervening DAS-MASK-EXPOSE operation and thus not removed during the DAS-PHOTORESIST-DEVELOP operation. However, tracing the functional and logical dependencies from the quantity representing the PMOS channel concentration might or might not lead to the hypothesis that the photoresist was not present during the implantation step.

This is due to the frame problem, the nature of logical reasoning, and the nature of implication. Recall that to overcome the frame problem, the system assumes that conditions persist through time unless it can prove that they are changed. In the nominal version of the manufacturing plan, the N-channel threshold adjust implant does not affect the concentration in the P-channel regions. Thus, the system merely records that the dopant concentration in those regions persists through the step just as it persists through the photolithography, etching and some other steps. The dependency network does not keep records of what it cannot prove.

The law of the excluded middle states that every well-formed statement must be either true or false. However, in reasoning the question that is settled is not whether statements are true or false, but whether they are entailed by (actually, provable from) the truth or falsehood of other facts. In reasoning there is a third truth-value: unknown. The ion-implantation model will actually form statements about changes to the dopant concentration of every layer-region. If these statements can be proven true, then these changes and the facts that they depend on are recorded. If they can be proven false, then the logical dependencies leading to those conclusions are also recorded. However, for those layer-regions that are not normally affected, it will often be the case that nothing will be proven concerning whether they are changed, and the persistence assumption will prevail.

A major reason that nothing will be proven is that the models are often based on implications. That is, statements of the form A implies B, where A is a statement about the conditions of processing, and B is a statement about the effects of that processing. If the antecedent of the implication, A, is proven true, then the truth of the conclusion B follows. However, if the truth of A is either false or unknown, then the truth of B is not determined, and therefore unknown.

By tracing the dependency network, one can directly answer the question: what factors lead to the truth or falsehood of statements whose truth has been determined, and how do the values of quantities functionally depend on other variables. To arrive at a diagnostic hypothesis involving a topological change, one must often answer a different question: what changes in the truth of statements would lead to a proof that an unexpected change occurred? Answering this type of question can involve an open-ended search of possible counter-factuals.

If the topological change had been simulated, and the consequences of that change had been labelled with their logical dependence on the assumption of the topological change, then such hypotheses can be uncovered by straightforward search of the dependency graph. However, this requires that such topological faults be anticipated.

Thus, even though they treat phenomena that lie within the theory represented by the models, causal associations in the second category make explicit a crucial piece of knowledge: that certain topological-change hypotheses are worth considering. They also

make it possible to treat such problems without actually simulating the topological change.

The criteria for instances of the generic rule for this causal association are:

1. that a layer-region unaffected by an implant step is in a region that was covered by photoresist during the implant step;
2. that the layer-region would have been affected if the photoresist had been absent;
3. that the photoresist was patterned by a particular photomasking operation;
4. that the photoresist was placed on the wafer by a particular coat operation.

#### 7.1.1.5. Category 3: Associations outside scope of theory

Generic Rules for causal associations in the first two categories facilitate diagnosis for phenomena that are encompassed by the theory. Causal associations in the third category go an important step further: they enable diagnosis of problems involving phenomena that lie outside the competence of the theoretical models. In Chapter 6 we discussed the Poly-Gate-Under-Etch causes Channel-Length-Long causal association. It was explained there that this association involved lateral etching and therefore fell outside the competence of our one-dimensional models. Variations of this particular association and a similar one regarding the exposure of the photoresist masking the etch account for 8 of the 12 category 3 causal associations in the AESOP knowledge base. The remaining four causal associations in the third category are variations on the following:

```
Gate-Ox-Time-Long → P-Ch-Conc-High (Very-Likely)
```

This associates the length of time that the gate oxide is grown with the concentration of dopant in the channel of the PMOS device. The phenomenon at work here involves subtle interactions between oxidation and diffusion which, though fairly well understood, are not captured in our discrete action models due to the extremely coarse representation of dopant profiles, diffusion and oxidation that we employ. Specifically, when doped silicon is oxidized, the dopants at the moving interface between the growing oxide layer and the silicon layer can either enter the oxide layer or stay in the silicon layer. P-type dopants such as boron are more soluble in silicon dioxide than in silicon, and therefore they tend to enter the oxide layer. This depletes the silicon layer of p-type dopants. N-type dopants like arsenic prefer to stay in the silicon layer, and therefore “pile up” in that layer. The net effect is that the oxidation makes the concentration of dopants in the silicon layer more n-type and less p-type. If the

dominant dopant is n-type, the effect is to increase the concentration, and if the dominant dopant is p-type the effect is to decrease the concentration.

### 7.1.2. Testing Diagnostic Knowledge Against Faults

To test the effectiveness of the diagnostic knowledge, we must have factory data concerning the manifestations of manufacturing problems that were deliberately introduced. It is necessary that the problems be deliberately introduced in order to have reliable ground truth regarding the correct diagnoses.

Obtaining this data would normally require pushing wafers through the fabrication facility with several modified versions of the manufacturing process. This would have to be done very carefully to avoid introducing faults other than the ones we intend. Such an experiment would take considerable amounts of time, and expertise that I do not possess. Fortunately, some experiments of this nature have already been performed at Stanford by Greg Freeman [5] .

These experiments were performed to evaluate a knowledge-based diagnostic tool called MERLIN. MERLIN employs symbolic and quantitative reasoning with analytic equations to diagnose faults in device behaviour and device characteristics as faults in device structure. Our focus is on diagnosing faults in device structure as faults in processing. Thus, our work is complementary to that of Freeman. It should enable the diagnoses performed by MERLIN to be carried to the next stage.

Thus, we test the knowledge obtained from the first experiment described in section 7.1.1 by attempting to explain the device structure faults found by MERLIN in terms of processing faults.

Freeman performed seven experiments. Six of the experiments involved systematic structural perturbation, with data taken from the Stanford BiCMOS process. The seventh involved a random structural integrity problem using data taken from the literature. The causal dependencies generated by assimilation of a process with our discrete action models, and the causal associations in the AESOP knowledge base are relevant only to systematic perturbation problems: they are not competent to deal with random integrity issues. Of the six systematic perturbation problems, five concerned the N-channel MOS transistor and the sixth concerned interconnect. Our assimilation of the process stopped short of dealing with the interconnect layers, and our models would not

be useful in treating interconnect problems. The AESOP knowledge base also lacks any associations dealing with interconnect problems. Thus, only the first five of Freeman's problems are relevant for our evaluation.

The data for these experiments was obtained by running wafers through the BiCMOS process with splits that intentionally introduced perturbations in the processing conditions. His five experiments actually only involved three perturbations. The perturbations introduced were:

1. reduced temperature during the source/drain drive-in;
2. increased energy in the NMOS threshold adjusting implant;
3. combination of: reduced source/drain drive-in temperature, increased gate oxidation time, and reduced threshold implant dose.

For each of the first two perturbations in processing, two different scenarios regarding the availability of data were investigated. Problem 1a involved comprehensive data from a single die. Problem 1b involved only a subset of the data for a single die. Problem 2a again involves a relatively complete dataset from a single die that enables verification to be performed. Problem 2b involves data averaged over all the die on a wafer.

## 7.2. The Results

### 7.2.1. Assessing Generic Rules

The 52 causal associations in the AESOP knowledge base can be grouped according to the criteria for valid instantiation that they share. Thus, only 15 generic rules suffice to cover the causal associations. These rules are listed in Appendix E. The rules were applied against an extremely simplified representation of the CMOS process to determine whether they would reproduce the causal associations that inspired them. The resulting causal associations are presented in Appendix F. That appendix lists the associations in a "raw" form, before a concept classifier has been applied. Then the generic rules were applied to the simulation of the manufacture of three transistors side-by-side (an NPN bipolar transistor, an NMOS transistor and a PMOS transistor) using the BiCMOS process as extracted from the Specification Editor knowledge base. The subsections below discuss the recovery of the original AESOP causal associations, and the associations that were obtained from the BiCMOS process.

#### 7.2.2.1 Category 1 Causal Associations

The first rule, entitled Ox-Temp-Determines-Ox-Thickness, handles the first eight of the 34 category 1 associations. It is instantiated whenever there is a layer of oxide created during an oxidation step. When applied to the simplified CMOS process, this rule instantiates all eight causal associations directly. As shown in Appendix F, 16 causal associations are generated. They represent the same eight causal associations that were present in the AESOP knowledge base, except that each is repeated twice because the Gate-Oxide-Layer is represented by two distinct layers in two different regions of the wafer (recall our comments in Chapter 4, Subsection 4.4.2 regarding the fact that the models sometimes make distinctions not made by semiconductor technologists). As each layer is named Gate-Oxide-Layer, a concept classifier could resolve these 16 associations into the requisite eight.

Note that the rule only found associations concerning the gate oxidation. However, this is due to the fact that our simplified CMOS process has only one oxidation step: for a more realistic process instances would be found for every oxidation step in the process. Indeed, this is what happens when the rule is applied to the BiCMOS process: 144 instantiations are found, representing eight rules for each instance of an oxidation creating a layer. The eighteen pairings of oxidation steps with layers created involve ten oxidation steps. Two oxidation steps are paired with two layers each, and three are paired with three layers each. All these pairings are correct. However, there are two key facts to point out. First, the rule was written to instantiate only if the oxide layer was “created” by the oxidation step, where this means that a new oxide layer is created where none existed before. Thus, when an oxidation step merely increases the thickness of an existing oxide, no instantiation occurs. This represents a problem for the Field Oxide case. The Field Oxide is a very thick oxide that is grown from a relatively thin oxide layer called the Pad Oxide. Thus, it is a significant oxide layer that is created more by the second oxidation step affecting it than by the oxidation step that created the Pad Oxide. A more sophisticated rule would consider all the oxidation steps that affect each layer, and instantiate the rule for the “most significant” step, i.e., the step that most contributed to the thickness of the layer. Secondly, oxidation steps often create oxide layers that are “insignificant” and that are later stripped away. For example, the oxidation step that grows the Field Oxide actually creates a very thin oxide layer by oxidizing the Silicon Nitride layer that masks the growth of the Field Oxide.

The rules Implant-Dose-Affects-Concentration-1, Implant-Dose-Affects-Concentration-2 and Implant-Dose-Affects-Concentration-3 handle the associations concerning implants into doped layers. The first rule handles the case when the implant element type is the same as the dopant in the receiving layer, the second handles the case when the types are opposite and the implant is not expected to change the dominant dopant type and the third rule handles the case when the implant is expected to change the dominant dopant type. These three rules cover the associations concerning the effects of N-Vt-II and P-Vt-II on the P-Ch-Conc and N-Ch-Conc. There are 12 of these in the AESOP knowledge base.

When applied to the CMOS process, the generic rule instantiations recover 48 associations. The first rule generates 16 associations, the second generates eight associations, and the third produces 24 associations. The four associations concerning the effect of the N-Channel implant (N-Vt-II) on the N-Channel concentration (N-Ch-Conc) are recovered directly by the first generic-rule. The four associations regarding the effect of the P-Channel implant (P-Vt-II) on the P-Channel concentration (P-Ch-Conc) are also recovered directly, by the second generic rule. The four associations regarding the effect of the P-Channel implant on the N-Channel concentration are each captured by two associations generated by the first rule: one association relates the P-Channel implant dose to the concentration of Layer-3, which is the layer created in the NMOS regions by the P-Channel implant; the second relates the concentration of Layer-3 to the N-Channel concentration.

The remaining four associations found by the first rule concern the impact of the substrate concentration on Layer-3, and thus ultimately on the N-Channel concentration. The AESOP knowledge base lacks corresponding associations. The remaining four found by the second rule relate the concentration in the N-Well to the concentration in the P-Channel. These represent half of the pairs of associations that would together account for the effect of the N-Well implant on the P-channel concentration. The other four associations needed, concerning the effect of the N-Well implant on the N-Well concentration, are found by the third rule. This rule also generates four associations regarding the effects of the substrate concentration on the P-Channel concentration (via its effect on the N-Well concentration).

The remaining 16 associations found by the third rule concern the effects of the P+ and N+ Source/Drain implants on the sources and drains of their respective

transistors, and the influence of the channel concentrations on these layers. The AESOP knowledge base also lacks these associations.

When these three rules are applied to the BiCMOS process, the first rule generates 24 associations, the second rule generates no associations, and the third rule generates 40 associations. The 24 associations found by the first rule concern the impact of the implant dose for each of the Channel-Stop, P-Channel and N-Channel implants on the layers created by each of them, and the impact of the Substrate and N-Well concentrations on these layers. No associations are found by the second rule because, unlike the CMOS process, the BiCMOS process has no implants involving an implant species of opposite type to that already in the layer to be implanted, for which the implant is not expected to change the dominant dopant type. The 40 associations found by the third rule concern the effect of the implant dose on the implanted concentration for each of the N-Well, Collector, Active-Base, N+ Source/Drain and P+ Source/Drain implants, and the impact of the initial concentrations (Substrate, Substrate, Collector, N-Channel and P-Channel respectively) on the resulting concentrations.

The four AESOP associations concerning the impact of the initial substrate concentration on the final substrate concentration are recovered directly by the generic rule Initial-Doping-Affects-Final when this rule is applied to the CMOS process. The corresponding four associations are also found when the rule is applied to the BiCMOS process.

### 7.2.2.2 Category 2 Causal Associations

The six causal associations in this category in the AESOP knowledge base were to be recovered by four generic rules applied to the CMOS process. The first association in this category, regarding omission of the Gate-Oxide step, is recovered directly by the generic rule Oxidation-Step-Omitted. Applying this rule to the BiCMOS process generates 13 associations regarding the Pad Oxide, Gate Oxide, Poly Frame Oxide, Sidewall Oxide and the Resistor Oxide. The rule Initial-Dopant-Type-Affects-Final directly recovers the causal association regarding incorrect initial dopant species in the substrate (although it is expressed in terms of dopant type rather than species) when applied to either process.

The generic rule Implant-Step-Omitted recovers the two associations regarding omission of the Well implant and incorrect species for this implant. It also generates eight associations regarding the two Source/Drain implants that are missing from the

AESOP knowledge base but seem perfectly reasonable. Applying this rule to the BiCMOS process generates 14 associations regarding the N-Well, Collector, Active Base, N+ Source/Drain and P+ Source/Drain implants. (The latter two implants create four rules each, as they also affect the polysilicon layer.) All these associations are correct.

The generic rule Implant-Not-Masked was intended to recover the association regarding omission of the Well implant. When applied to the CMOS process, this rule recovers the association regarding the N-Well implant, but also generates two spurious associations concerning the effect of the two Source/Drain implants on their respective channel layers, and two correct associations regarding the impact of not masking these implants on the polysilicon layer. The spurious associations are in fact due to an inappropriate arrangement of mask regions for the channel implants, i.e., a failure on my part to correctly represent the CMOS process. When applied to the BiCMOS process, this rule also generates five associations. However, the spurious associations are not present. The five associations found are the three correct associations corresponding to the CMOS associations, and two more concerning the Collector and Active Base implants.

This rule does not recover the association regarding failure to mask the N-Channel implant because it is restricted to cases where the implant changes the dominant dopant-type. An additional rule is required to handle this case.

#### 7.2.2.3 Category 3 Causal Associations

The four associations relating over- and under-etching of the polysilicon gate to the length of the channel are covered by twelve associations created by two generic rules: Insufficient/Excess-Lateral-Etch and Structure-Masked-Implant applied to the CMOS process. The first rule generates eight associations that establish that over- or under-etching the gate affects the width of the gate itself, for each of the two MOS devices. The second rule produces four associations that establish that the width of the gate affects the width of the channel due to self-aligned implantation steps for each of the two devices. Precisely the same twelve associations are found for the BiCMOS process.

The generic rules Insufficient/Excess-Photoresist-Exposure and Resist-Masked-Etch generate interlevel and intralevel associations tying over- and under-exposure during the photolithography step that patterns the resist for the gate structure to the width of the poly gate. In this way, the four associations in the AESOP knowledge base

are each represented a by chain of three associations: exposure affects width of resist, width of resist affects width of poly gate, width of poly gate affects width of channel. When applied to the BiCMOS process, 36 instances of the Insufficient/Excess-Photoresist-Exposure rule are found. The Active Area mask creates three sets of four instances, one set for each of the three types of devices. Another set of four associations is found for the bracketing of the NMOS device by the Channel Stop mask. The Buried Contact mask produces two sets of four associations, one each for the NMOS and PMOS devices because the contact holes bracket the gate oxide. The Poly Oxidation mask generates four associations for the way the Poly Frame Oxide brackets the NPN Emitter region. Finally, two sets of four associations are found for the Poly Etch mask, which correspond to the definition of the Poly Gate structures in the NMOS and PMOS devices. The Resist-Masked-Etch rule finds all the same situations (except the Channel-Stop case, which does not involve an etch), but produces an even larger number of associations (48), because in some cases more than one layer is etched.

The generic rule Thermal-Oxidation-Redistributes-Impurities-N correctly recovers the four AESOP causal associations regarding the effect of extended gate oxidation on the concentration of the P-Channel concentration. In addition, a companion rule Thermal-Oxidation-Redistributes-Impurities-P adds four associations regarding the impact on the N-Channel concentration that are missing from the AESOP knowledge base. Applying these rules to the BiCMOS process generates a plethora of associations: the first rule finds 88 associations (22 groups of four), and the second finds 72 (18 groups of four). As I interpreted the underlying cause for the AESOP rule (i.e., oxidation surface effects), these associations are all correct. However, I most likely misinterpreted the physical mechanism underlying the causal association. This is discussed in the section on “Expert Assessment.”

Table 7.1 summarizes the results regarding recovery of the AESOP causal associations by application of the generic rules to a simplified representation of the CMOS process. All but one of the 52 associations in the AESOP knowledge base are effectively represented. In addition 30 more causal associations are found that might prove useful. Only six spurious associations are generated: 2 due to poor representation of the process, and 4 due to misinterpretation of the physical mechanism underlying an AESOP association.

| Category | Recovered | Missing | Additional, Correct | Additional. Spurious |
|---|---|---|---|---|
| 1 | 34 | 0 | 20 | 0 |
| 2 | 5 | 1 | 10 | 2 |
| 3 | 12 | 0 | 0 | 4 |
| total | 51 | 1 | 30 | 6 |

Table 7.1. Recovery of AESOP associations by Generic Rules

Applying the same 15 generic rules to the BiCMOS process results in over 500 causal associations. The definition of the Ox-Temp-Determines-Ox-Thickness rule (and the manner in which the BiCMOS process constraints represent the creation of the Field Oxide layer) prevented a instantiations for the Field Oxide layer. However, this is the only significant set of “missing” causal associations. On the other hand, the phenomena in several AESOP causal associations were discovered to be active in the creation of the NPN Bipolar device in the BiCMOS process (these include causal associations concerning the impact of implant dose on dopant concentration, the impact of existing dopant on the concentration in implanted layers, the effects of omitting implant steps, and the effects of failing to mask implant steps, for which instances were found for the Collector and Active Base implants). Also, the phenomena reflected in some of the associations were found to be at work in implants into CMOS structures that were not considered by the AESOP knowledge base (i.e., the source/drain implants). The same expansion of cases occurs for associations regarding oxidation phenomena.

It is also the case that some AESOP causal associations were eliminated as inappropriate to the BiCMOS process. Specifically, associations concerning the impact of the P-Channel threshold adjust implant in N-Channel regions. Other associations were modified: the impact of the P-Channel implant on the P-Channel is the opposite in the BiCMOS process to what it was in the CMOS process, as the implanted dopant is the opposite type.

Finally, it should be noted that even when AESOP rules apply directly to the BiCMOS process, they cannot always be used directly because some naming conventions are different for the two processes (e.g., the threshold adjust implants of the CMOS process are called channel implants in the BiCMOS context). The instantiation process automatically adopts the naming conventions of the new process.

### 7.2.2. Expert Assessment

We consulted the principal designer of the Stanford BiCMOS process, John Shott, to obtain the judgment of an expert concerning the quality of the diagnostic information obtained by qualitative simulation and by application of generic rules,

A top-level comment concerned the lack of attention to issues of sensitivity: the knowledge as represented tends to be qualitative and therefore potentially too inclusive. For example, the AESOP knowledge base contained several causal associations regarding the impact of the concentration of dopant initially in a layer on the concentration resulting from an implantation into that layer (for example, Sub-Start-Conc-High → P-Ch-Conc-Low). This association stems from the fact that the dopants introduced into a layer add up (if they are opposite type, they subtract) to determine the ultimate concentration. The DAS qualitative simulator and the generic rules concerning this phenomenon ignore the relative sizes of the quantities being added, and assume that a change in any of them results in a change in the sum. However, the concentration of dopant introduced by an implant will often be one or more orders of magnitude higher than the concentration already present in the layer, so that the impact of variations in the existing concentration can be quite small. As a rule of thumb, implants that are intended to change the dominant dopant introduce at least an order of magnitude more dopant than is present in the layer to be implanted.

In a similar vein it was noted that some variations in geometric or other physical attributes are not interesting unless they are large enough to produce catastrophic results. For example, while the causal associations relating to lateral dimensions of the buried contacts are correct, device and circuit performance are largely insensitive to such variations.

The need for quantitative analysis is reflected in another way. The HyperPIES causal associations include a qualitative estimate of likelihood. It is not clear whether this represents the likelihood that the manifestation will occur if the problem occurs, or the likelihood that the problem is the correct explanation of the manifestation. (The original PIES system included two assessments of likelihood, but this approach was dropped when the system was ported to HyperClass.) Currently, the generic rules merely carry these likelihood assessments as given. This leads to two problems. First, when a single causal association is represented by a chain of generic rules, it is not clear how the likelihood should be distributed. Second, the author of a causal association

would probably assess likelihood differently for each instance of a corresponding generic rule. The author's assessment would probably depend on factors determining the sensitivity of the phenomenon, and on subjective probability densities for the kinds of failures being considered. It is not clear how all the factors that influence a subjective likelihood assessment could be addressed. However, to the extent that the experts can enunciate how their assessments depend on aspects of the situation, it should be possible to record and make use of this information.

That information concerning intermediate states is represented was seen as an advantage. In addition to the benefits of a richly interconnected causal network I have already described in previous chapters, breaking an input/output relationship down into several intermediate stages is useful because it is sometimes possible to make measurements of these intermediate states. Fabrication facilities often routinely make and record such "in-process" measurements. However, a diagnostic system that does not represent such intermediate states cannot incorporate these measurements into its strategy for discriminating among otherwise indistinguishable cases.

For similar reasons it was seen as an advantage that the system systematically includes consideration of "unimportant" and "sacrificial" structures. For example, when growing the field oxide and when oxidizing the polysilicon layer in the BiCMOS process, a very thin oxide is simultaneously formed on top of the silicon nitride that is used to pattern these oxidations. This oxide is an unintended side effect that is removed very shortly afterward. It is not part of the final structure being manufactured and it plays no significant processing role. It is thus easily ignored or forgotten. However, it can be a source of problems, especially when its presence is overlooked. Also, some oxides are deliberately grown only because doing so enhances a diffusion process. These oxides are also etched away. However, their thickness is often measured as an indirect means of establishing that the diffusion processing occurred as planned. Such measurements are potentially useful diagnostic information.

Only one phenomenon was a source of surprise. The generic rules concerning the surface effects of oxidation did not seem credible. Recall (Section 7.1.1.5) that the AESOP rules relating the duration of the gate oxidation step to the concentration of the P-Channel were assumed to involve the segregation of dopants into either the oxide or the silicon layer, depending on dopant type. Normally, the impact of any high-temperature

step is to reduce the concentration of dopants. Thus, these surface effects were invoked to explain how an extension of an oxidation step could increase dopant concentration.

This explanation lacks credibility because these surface effects are generally "second order" effects that are swamped by the first order effect of oxidation and diffusion, which is to reduce dopant concentration. This led to another possible explanation for the phenomenon at work in this causal association, in terms of first order effects. Namely, if a layer has both p-type dopant and n-type dopant, and the p-type dopant diffuses faster than the n-type dopant, then the concentration of the p-type dopant will decrease faster than the concentration of the n-type dopant, and the layer will become more n-type. The p-type dopant can diffuse faster than the n-type dopant if it has seen less high-temperature processing (so that its concentration gradient is higher), or if the n-type dopant is Arsenic, which diffuses ten times more slowly than the p-type dopant Boron. In the CMOS process, both conditions hold: the N-Well is formed by an Arsenic implant, which is driven-in before the P-Channel implant is performed.

The generic rules do not explicitly refer to these secondary surface effects. However, if the mechanism responsible for the original association is the first-order effect just described, then the conditions under which the phenomenon will occur are more restrictive. Most instances of the current rules would not satisfy these restrictions.

### 7.2.3. Assessing Diagnostic Capability

The decision to employ data from a set of experiments performed completely independently of this work was taken for two reasons: to avoid the possibility of biasing the evaluation by choosing problems that the system can handle easily, and to make it possible for the evaluation to employ real factory data. This decision has led to a sparse evaluation that is probably not a fair assessment of the system's value.

The data from the five experiments performed by Greg Freeman really only represent three experiments. The main distinction between experiments 1a and 1b, and between 2a and 2b is the quality of the electrical measurement data. The input to our diagnostic process is the output of the MERLIN diagnostic process. This output does not strongly reflect the differences in the quality of the input data. For experiments 1a and 1b, the only distinction is that a second hypothesis is generated that cannot be distinguished from the correct hypothesis. For experiments 2a and 2b, the same answer is obtained, with only a change in the confidence associated with that answer.

The generic rules that we created were motivated by the causal associations in the AESOP knowledge base. With one exception, those causal associations are not relevant to the problems treated by Freeman. Thus, our BiCMOS causal associations, generated from those generic rules are not relevant to the problems either. This means that the problems can only be used to evaluate the dependency structure created by qualitative simulation.

#### 7.2.3.1 Experiments 1A and 1B: Low temperature Source/Drain Drive-in

MERLIN rejects all but one hypothesis for experiment 1a, and all but two for experiment 1B.

The winning hypothesis in experiment 1A is “DELTA_LD.” This is Freeman’s term for a variation in the lateral diffusion under the gate. As this is a lateral diffusion phenomenon, it lies outside the phenomena scope of the DAS models! However, we can proceed by noting that diffusion is completely anisotropic, so that the amount of lateral diffusion is reflected in the amount of vertical diffusion. Asking the DAS dependency network for the processing parameters influencing this quantity yields a response that includes the duration and temperature of the drive-in step, as well as the factors that determine the implanted concentration and the concentration of the channel-layer (these are included because the diffusion rate is partly determined by the gradient of the dopant profile, which in DAS is represented by the difference between the concentrations in the adjacent layers). Thus, the correct answer, a lower drive-in temperature, is included in the response, but it is one among many.

Additional data, if it were available, could be used to filter this set of hypotheses. For example, normal measurements of the sheet resistance of the polysilicon wires could indicate that the implanted concentration was not abnormal. Also, in-process measurement data might have indicated that the channel concentration was correct. Assuming such data were available, the system could eliminate all hypotheses except for the temperature and duration of the drive-in step. These two hypotheses could not be distinguished by DAS, because it does not have quantitative models of the relative influence of time and temperature on the oxidation rates. However, the hypotheses could be distinguished on the basis of measurements of the oxide thickness if such information were available.

The hypotheses returned by MERLIN for experiment 1B are “DELTA_LD” and “DELTA_LG.” This second hypothesis refers to the difference between the gate dimension as drawn, and the physical dimension. The BiCMOS version of the AESOP rules concerning insufficient lateral etching of the gate, and under exposure of the resist patterning the gate come into play here. These add additional hypotheses relating to the duration of the Poly Etch step and to the exposure parameters for the Poly Etch mask step. However, since they would also indicate the intermediate hypothesis that the physical dimension of the polysilicon gate was too long, a diagnostic reasoner could try to eliminate these hypotheses by an optical measurement of the actual gate length.

#### 7.2.3.2 Experiments 2A and 2B: high energy channel implant

For these problems, MERLIN returns the hypothesis “D_TI,” which is Freeman’s term for depth of the threshold implant. The phenomenon at work here is that the dopants expected to modify the threshold voltage are implanted too deeply, so that the concentration at the silicon surface is too low.

The constraints attached to the channel (= threshold adjust) implant indicate that the region implanted lies at the silicon surface. Thus, this phenomenon involves a topological change, introducing a layer between the silicon surface and the implanted layer. There are two ways to continue the diagnosis to the process anomaly stage: one way is to simulate the possibility of a threshold implant that lies beneath the surface, then ask what factors would determine the actual depth. The other way is to have a generic rule that relates this phenomenon to the mean implant depth (and thus to the initial thickness of the implanted layer), and ask what factors determine that. In either case, the result is the same: the hypotheses returned include the implant-energy of the channel implant, as well as the factors that determine the thickness of the Cleanup Oxide.

The factors determining the thickness of the Cleanup Oxide are included because the channel implants are done through this oxide layer, and the thickness of the layer would help to determine the implant depth. As the thickness of an oxide over silicon is relatively easy to measure, and such thicknesses are routinely measured just after the growth of the oxide (the BiCMOS process description includes such measurements), these additional hypotheses might be eliminated by normal oxide thickness measurements.

#### 7.2.3.2 Experiment 3: Multiple Faults

Freeman performed one experiment in which the same wafers were subjected to three process disturbances: lower temperature source/drain drive-in, longer gate oxidation and lower threshold implant dose. Unfortunately, the principal purpose in performing this experiment was to demonstrate a weakness in MERLIN's diagnostic strategy, which assumes a single point of failure. Thus, MERLIN incorrectly reports the physical manifestation ("DELTA_LD") of only one of the three problems (low source/drain drive-in temperature).

Thus, the input to the next stage of diagnosis for this problem is the same as it is for problem 1A, and DAS would return the same result.

## 7.3. Summary of the Results

It has been demonstrated that the theoretical models can be used to assimilate a new, complex manufacturing process by qualitative simulation. The procedure is computationally expensive, both in terms of time and space. However, this cost can be amortized over the large number of diagnoses that the resulting dependency structure can be used to perform. The causal dependency network includes many interior nodes representing intermediate states of the wafer. This improves the discriminating power of the network by providing many opportunities for seeking contradicting and confirming information. In particular, in contrast to statistical input/output techniques, it affords the capability to employ information from in-process and in-situ measurements.

A key weakness of the approach stems from its lack of quantitative information. This reduces its ability to discriminate on the basis of sensitivity, leading perhaps to an excessive number of hypotheses.

It has also been demonstrated that the dependency network can be used, in conjunction with generic rules, to transfer experiential knowledge from one context to another. To avoid creating spurious causal associations, it is important to identify and properly encode as many as possible of the criteria restricting when phenomena can be expected to occur. On the other hand, by capturing the description of the phenomena in as general a fashion as possible, one reaps the benefit of additional knowledge when new instances of the phenomena are discovered.

# Chapter 8. Related Work

In this chapter we review other work related to the subject matter of this dissertation. Where possible we note either how our work represents an advance in the state of the art, or how our work might be improved by incorporating the ideas contained in these other efforts. We begin with a brief review of the application of model-based reasoning techniques in other domains. Then we contrast Generic Rules with other approaches to combining rule-based and model-based reasoning. Next we examine other computational tools for modeling semiconductor manufacturing. As this work touches on issues in semiconductor process representation, we briefly describe other efforts in this area. Finally, we discuss other approaches to the problem of deciding how semiconductor manufacturing operations should be modeled for a specific application.

## 8.1. Model-Based Reasoning

One of the ways we attempt to achieve robust encoding of knowledge about semiconductor manufacturing in this thesis is to decompose manufacturing plans into their constituent parts, namely manufacturing steps, and encode knowledge about those parts in a fashion that enables us to reason about plans composed of them. To admit composition into manufacturing plans, the knowledge we encode about each step must have a general character: it must apply in all occurrences of the step within plans. For this reason, the knowledge tends to be theoretical in nature, and to constitute a model of the step's behaviour.

This idea is by no means new. Model-based reasoning techniques have been applied in a variety of domains. The earliest symbolic model-based reasoning systems were applied in the domain of electronic circuits — primarily digital circuits, but also analog circuits — (e.g. [20, 52-55] .) It has also been applied in medical domains (e.g. [16, 56-59]). More recently, the techniques of qualitative reasoning have enabled the application of model-based reasoning to many types of engineered systems [60], including civil engineering structures (e.g. [61]), power plants (e.g. [62]), chemical plants (e.g. [63, 64] ), aircraft (e.g. [65, 66] ) and spacecraft (e.g. [67-70]). In the semiconductor

domain, it has been applied to reasoning about material surfaces and interfaces (e.g. [71]). The techniques are now also being applied to financial domains (e.g. [72-75]).

In all cases the structure of the system being reasoned about is represented by a graph in which the nodes are components of the system and the arcs are the conduits of communication or interaction between the components. The main distinctions between our work and these concern the complexity of the signals that pass along the conduits between parts of the system structure, and the treatment of time. Both of these distinctions derive from the nature of the domain about which we are reasoning. As noted in section 4.2 of Chapter 4, the components of a manufacturing plan are the steps in the plan, and the signal that passes between these components is the description of the object that the manufacturing steps operate on. This is a highly structured object with an *a priori* undetermined number of components. In almost all previous work the signals that pass between components consist of scalar quantities. Furthermore, in much of the work the number of such quantities is determined by the structure of the system being reasoned about and is known at the outset of reasoning: i.e. no new quantities are discovered in course of reasoning. As we mentioned in Chapter 4, the process-centered reductionist techniques are an exception to this rule [18]. In these techniques new quantities can be introduced by the discovery and activation of instances of processes. However, it is often the case that systems based on this approach perform an *envisionment*. This procedure identifies all potential instances of processes at the beginning of reasoning, assimilating the structure of the system being reasoned about in much the way we assimilate the structure of the manufacturing plan.

Regarding the treatment of time, previous work falls into four broad categories. The first category ignores time altogether: the system being reasoned about has no state. It is characterizable by a set of algebraic equations [76]. Much of the early work on digital circuits considered only combinational logic, and fell into this category [3, 20, 21, 52-54]. The second category considers time as quantified into uniform intervals, with discrete changes taking place instantaneously at the end of each interval. The prototypical example of this is synchronous (i.e. clocked) digital sequential systems. The third category treats dynamical systems. These can be characterized by sets of first-order ordinary differential equations [17-19, 77, 78]. The scalar quantities change continuously, and the evolution of the system is often determined by feedback loops [79]. The fourth category allows for discontinuous change as well as continuous change in the values of the quantities [80]. Some work in this area also attempts to reason simultaneously at

several different time-scales, treating the same changes as discontinuous at one scale, and continuous at another scale [81].

Our work is based on that of Reid Simmons [14], which brings together ideas from two disparate areas of artificial intelligence: qualitative model-based reasoning and planning. The treatment of time within our system is that associated with reasoning about plans. Changes occur only as a result of actions. Although it may be recognized that the actions take place over an interval of time and some of the changes are actually continuous changes in the values of quantities, actions are considered to be atomic. The system considers an action as a direct mapping between the states of the world before the action and after it, but allows for the possibility that the mapping involves algebraic equations over continuous variables, with the duration of the action as an explicit variable. Furthermore, we do not consider the possibility of concurrent actions. Thus, our view of time is a very simple one. It can also be viewed as a hybrid of the first and second ways of treating time mentioned above: time is discretized into intervals, with all changes considered complete at the end of each interval, but the intervals are not uniform in duration. The values of quantities after each interval are continuous algebraic functions of the quantities in the prior world-state and the interval's duration. Discrete changes that introduce or remove objects are also possible.

Most semiconductor processing involves continuous change. Furthermore, the complexity of some processes, such as diffusion, make it awkward and difficult to describe them in a general way as discrete actions. Thus, it would seem that abstractions allowing for continuous change, concurrent processes and feedback would be appropriate for reasoning in greater detail about individual processing steps. However, current techniques are not completely adequate: the physical processes in some semiconductor operations can't be characterized by ordinary differential equations. They require *partial differential* equations, which lie outside the realm of current techniques. There may be some useful intermediate level of abstraction for which the ordinary differential equations that current techniques can handle are adequate. There has already been some work in this direction. In [82] qualitative and semi-qualitative continuous models are employed to aid in the diagnosis of individual processing steps (in particular, a plasma enhanced chemical vapor deposition (PECVD) of silicon nitride). Qualitative continuous models have also been used to model equipment and individual manufacturing steps in other manufacturing domains for the purpose of monitoring and control [83-85].

Despite the difficulties involved in representing semiconductor processing steps as discrete actions, this is the most appropriate level of abstraction for our purpose of unraveling all the interactions among manufacturing operations.

## 8.2. Combining Rule-Based and Model-Based Reasoning

There have been other attempts to combine rule-based and model-based reasoning. In most cases, the phenomena treated by each form of knowledge has been the same. The large inferences sanctioned by high-level rules have been viewed as contributing to fast reasoning. The detail with which model-based techniques handle phenomena has been viewed as contributing to robustness, and/or the ability to resolve interactions between diagnostic hypotheses [86, 87] .

Although we also distinguish between rules and models in our work, the key distinction we wish to draw concerns the content of the knowledge represented. In particular, the different forms of knowledge are distinguished by level of approximation, granularity, abstraction and the degree to which the knowledge is heuristic. It is our argument that by encoding heuristic knowledge at a high level of abstraction and granularity, rule-based experiential knowledge bases can treat a larger range of phenomena than it is possible to model effectively. The second part of our argument is that to encode the dependency of that knowledge on various aspects of context that are subject to change, a finer degree of granularity and a lower level of abstraction are required.

It has been suggested that increased reasoning speed can be achieved by "compiling" the knowledge contained in models into rules, perhaps designed for a specific task [67, 88-90] .

We agree with Randall Davis [91] when he stipulates that most of these efforts mistakenly emphasize form over content. As Tom Bylander points out [92], the "compiled" knowledge cannot be more competent for a given task than the knowledge it is "compiled" from, unless the "compiler" is endowed with knowledge that it can add during the compilation process. Also, if the compiled knowledge is to be more efficient at the task than the knowledge it is compiled from , then it must be the case that the problem-solving is an expensive, deliberative process with the knowledge in its original form. If that is the case, then the compilation process is necessarily also very expensive. It is Davis' argument that model-based diagnosis is not a deliberative process.

The work of Simmons [93] illustrates an effective way to combine the advantages of rule-based and model-based reasoning. In his Generate-Test-Debug paradigm, heuristic rules are employed to synthesize an interpretation of a geological cross-section as a possible sequence of geological processes. These rules recognize common configurations of geologic features and posit geologic sequences capable of producing them. Models of the behavior of geologic processes are then employed to test the resulting synthesized sequence by simulating it to determine what geologic cross-section would result, and comparing this with the original cross-section to be interpreted. Finally, the dependencies generated during this symbolic simulation are used to debug the sequence when discrepancies in the cross-sections are detected.

It should be noted that the heuristic rules employed in Simmons' work do not treat phenomena that are not already represented in the models he employs. Indeed, it is necessary that the models be capable of correctly simulating all phenomena that the rules discuss. One could imagine employing a general-purpose planner [30-32] to synthesize a sequence directly from the models, though this would entail a combinatoric search through a very large search space. The impact of the heuristic rules is to arrive at a reasonable estimate of the correct sequence without a great deal of search. Any attempt to compile these synthesis rules from the models would require that the compiler perform an exhaustive search of the space of possible sequences of geological processes.

Modern chess-playing systems are perhaps prototypical examples of systems that employ rule-based experiential knowledge, model-based theoretical knowledge and rule-based knowledge compiled from the model-based knowledge. In the early phases of a game, these systems employ an "opening book." This is essentially a rule-base with rules of the form: when in this situation, pseudo-randomly choose among these equally good moves. These rules are encodings of the accumulated experience of chess-playing experts. In the middle game, these systems employ model-based knowledge: the systems know what moves are legal, and what the effect of each move is on the board configuration. They employ this knowledge in a search of the space of possible move sequences that is pruned by heuristic knowledge concerning the "value" of each board position. Finally, in some systems there is also an "endgame book ." This, like the opening book, is a rule-base of best moves for each situation. However, endgame books are often generated by an exhaustive search using the model-based knowledge.

The issue in Simmons' work is *performance*: the efficiency with which each form of knowledge can perform its part of the task. While model-based knowledge is not expensive for diagnostic and debugging tasks, it is for the synthesis task. Performance is also the issue for most work in knowledge compilation. In contrast, our work is concerned with *competence*: the ability of the representational paradigm to capture the knowledge, and the degree of robustness that can be achieved. However, there are two ways in which the notion of knowledge compilation applies to this work.

First, our model-based knowledge (the models of the operations and the representation of the manufacturing plan) cannot be used to perform the diagnostic task directly with any degree of efficiency. Instead, it is necessary to assimilate the manufacturing plan through a qualitative symbolic simulation procedure. This procedure makes all the causal dependencies in the plan explicit. It is with this explicit graph of the causal dependencies that it is possible to perform diagnosis efficiently. The assimilation process is a transformation of the form of the knowledge into a form that is efficient for the diagnostic task. Furthermore, as in the case of most software compilers, the transformation involves considerable expansion in the size of the representation. Thus, the assimilation process can be viewed as a knowledge compilation process in which performance is improved by a computationally expensive process that makes explicit the knowledge that is only implicit in the original representation.

Second, we noted in Chapter 7 that it can be useful to have generic rules for phenomena that fall within the competence of the theoretical knowledge. It would seem that the effect of instantiating the generic rule in such a case is to merely change the form of the knowledge without changing its content. However, this is not entirely true. Even when the generic rule pertains to a phenomenon already captured by the model-based knowledge, the rule provides additional knowledge. Specifically, generic rules define the level of abstraction and granularity at which it is most appropriate to represent the phenomenon: what are the key concepts to be causally related, and what details can be omitted. Also, generic rules can contain heuristic knowledge in the form of certainty factors. Thus, the rule-instantiation process can be viewed as a knowledge-compilation process in which knowledge is added.

## 8.3. Modeling Manufacturing

In this work we have explored the symbolic approach to modeling semiconductor manufacturing in order to achieve robust representation of knowledge capable of

supporting diagnosis. There has been and continues to be much work focussed on other approaches to modeling semiconductor manufacturing.

Our models are very approximate and qualitative. To the extent that they include quantitative information at all, this quantitative information takes the form of simple analytic mathematical models. The work of Strojwas and others [37, 94] also employs simple analytic expressions to model semiconductor manufacturing operations. In that work, the expressions are tuned with empirical data to provide a fairly high degree of accuracy. The focus of that work is to provide quantitative simulation that is very fast, so that it can be used to support statistical experimentation and analysis. The simulator is called FABRICS, and is part of a larger suite of computational tools called the Process Engineer's Workbench.

While that work is similar to ours in its use of simplified analytic models of semiconductor manufacturing operations, it is aimed at a different purpose. The FABRICS simulator does not record logical and functional dependencies that can be traced to generate diagnostic hypotheses. Also, it cannot accept qualitative information — it is strictly quantitative. Its primary purpose is the statistical exploration of the input/output behavior of the manufacturing process in a small space around the points for which the analytic expressions were tuned with empirical data.

Early versions of the work reported in this dissertation inspired an effort similar to ours at Texas Instruments, led by Robert Hartzell [95]. As in the FABRICS work, the TI "deep level reasoning tool" employed quantitative analytic models embedded in code rather than our qualitative, declarative models. However, they structured the models with an object-oriented programming language in such a way that each model can include methods that record dependency information. In this way they attempt to achieve the best of both worlds: fast, quantitative simulation along with dependency information that can be used to support diagnosis. In our work, the dependency information is extracted automatically from the models, by virtue of reasoning with declarative statements in a logic and employing a truth maintenance system (TMS). In the TI work, the dependencies recovered for each operation are hand-crafted into a method attached to the model of the operation independently of the coding of the model itself.

This approach has advantages and disadvantages in comparison with our approach. Since the dependencies are hand-crafted, the level of detail and granularity in them can be carefully selected to match the needs of the diagnostic system. In our system the level of detail in the dependency network is determined by the demands of the simulator. This leads to a highly detailed dependency graph with nodes for every instance of every statement in each model. Also, since the TI models are compiled, quantitative programs, the assimilation of a new process plan is much faster than it is with our system. On the other hand, this approach places more of a burden on the programmer, especially with regard to maintenance. When the models are modified, the programmer must re-evaluate the coding of the dependency information. Also, the inability of the system to reason qualitatively with partially-specified information limits the utility of the models for other tasks that are carried out before a complete, quantitative specification is available.

Symbolic simulation of semiconductor processing is also employed in a system reported in [96]. This system checks newly designed manufacturing plans for violations of rules regarding safety and standard operating procedures. The system requires a representation of wafer state in order to check rules concerning constraints between wafer state and process or equipment. As in our system, it uses a rule-based simulator to create a representation of the wafer state at the start of each step from a representation of the manufacturing plan. However, their system uses representations of wafer state and processing steps that are even simpler than those we employ. They do not differentiate between different regions on the wafer surface. Instead, they consider that they need only keep track of what layers have been added to the wafer, and whether or not these layers have been patterned. Thus, their representation of the entire wafer is a simple stack of layers annotated with information regarding material, thickness, doping, and whether each layer has been patterned by a mask (or by a process that transfers a pattern between layers).

They are not concerned with extracting causal relationships between the manufacturing plan and the wafer state. Thus, their models of semiconductor operations are simple rules that directly specify the impact of the operation on their representation of wafer state. These simulation rules, furthermore, are quite specific to processing contexts. For example, one rule treats implantation into the polysilicon layer. This rule is only applicable if the wafer has a polysilicon layer exposed at the surface, and the

implantation energy is not sufficient to penetrate the wafer beyond that layer. Rather than a few general-purpose models, they employ about 50 specific simulation rules.

This work exemplifies the efficacy of symbolic reasoning with highly abstract models tailored to the needs of the task to be performed. It also confirms our position that an important use of computer systems is to manage the complexity stemming solely from the length of manufacturing plans: most of the errors found by the system are simple and obvious, but difficult for process engineers to spot due to the mass of detail and the distance in long process flows between steps that can interact.

There is much work going on — especially at Stanford, but also elsewhere — aimed at constructing highly accurate quantitative process simulators based on the best available knowledge of the physics and chemistry involved in the processes. These "physically-based" simulators use grid techniques to solve the systems of partial differential equations that model this physics [34, 35, 97-101]. The simulators operate at a very fine level of granularity and are consequently very computationally expensive. The focus in this work is to achieve a high degree of accuracy in simulation, and by being sufficiently comprehensive in capturing all relevant physical phenomena, to produce simulators that can reliably predict the behaviour of the processes over a wide range of operating conditions.

Figure 8.1 portrays some representative systems as "information transformers." The notion here is that these systems compute mappings between different levels of description. The sizes of the arrows in the diagram reflect the relative utility of these programs for transforming information in the forward and reverse direction. Quantitative simulators are best suited to computing these mappings in the forward, or *predictive* direction. For example, the SUPREM simulator computes a highly precise and accurate description of the state of the wafer after processing, from a description of the micro-environment created at the surface of the wafer during processing. Since simulators can only run in this forward direction, obtaining the inverse transformation is a trial-and-error process.

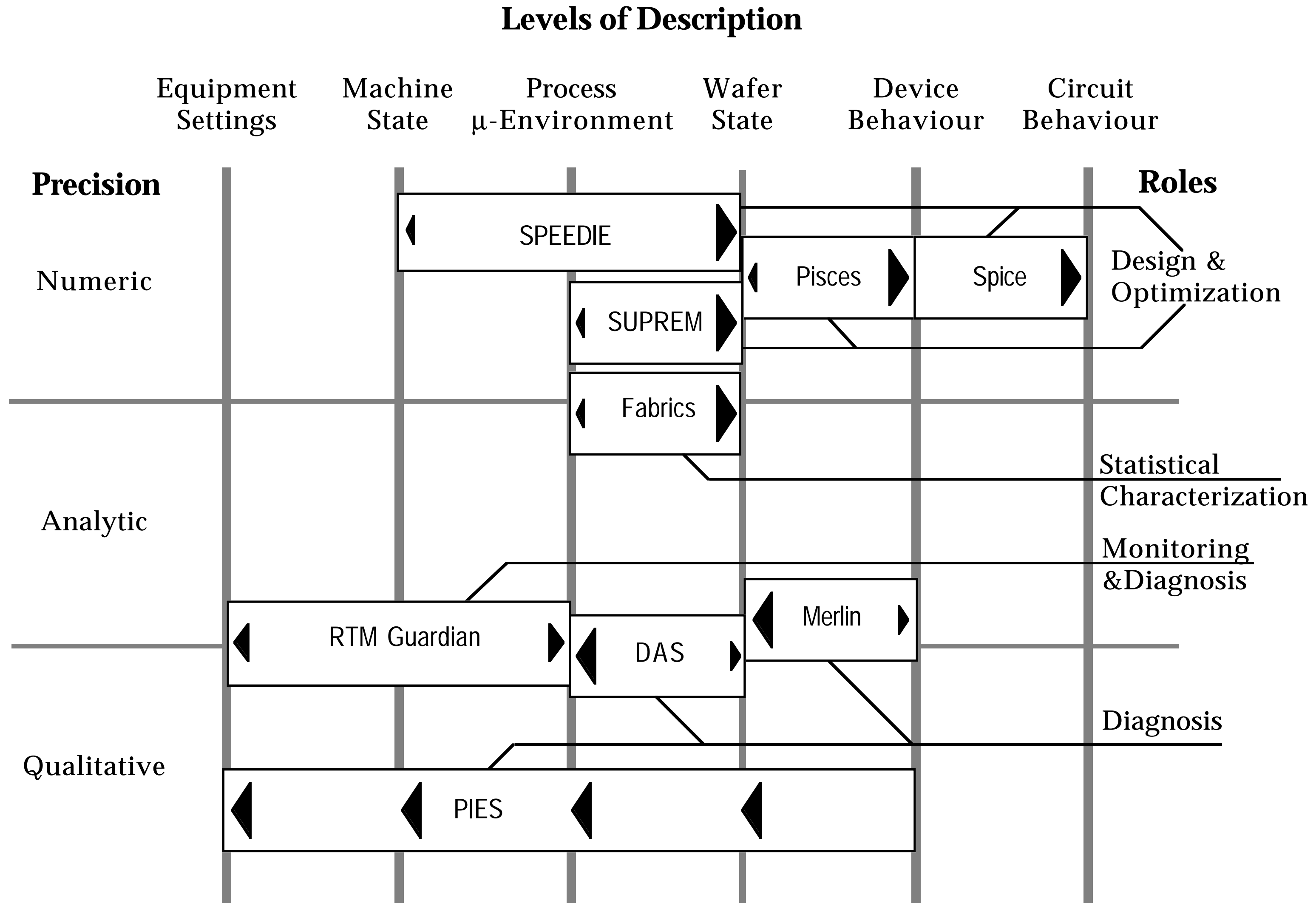


Figure 8.1 Semiconductor Computer Tools as Information Transformers

Computing the inverse transformation is best achieved with symbolic techniques. With DAS, for example, a forward simulation is performed to assimilate the manufacturing plan. However, this simulation is neither very precise nor highly accurate. Once the plan has been so assimilated, the dependencies recorded during simulation can be used to compute the inverse transformation: from a description of wafer state, determine the conditions of processing that led to it.

The complimentary character of the strengths of quantitative and symbolic techniques suggest a possibility for synergy. During diagnosis symbolic techniques can be used to generate hypotheses that can then be verified with quantitative simulation.

The AESOP work [11] represents another approach to achieving robustness in a causal-association based system. In this work, a set of statistical experiments were performed using the physically-based numerical simulators SUPREM and PISCES as a substitute for the real world. Response surface methods were used to characterize the overall input/output behaviour of the manufacturing process. From the resulting response surface, causal associations were extracted. Paradoxically, although the source of the knowledge is the theory embodied in the simulators, the content and form of the knowledge captured in the knowledge base is experiential: it encodes a set of experiences with the simulators, but doesn't explicitly encode any of the theory behind those experiences.

This illustrates another way to employ fine-grained theoretical knowledge to achieve robustness. The advantage of this approach is that it is based on the best theoretical knowledge available in computer-encoded form. As we pointed out in Chapter 4, mathematical models solved by numerical techniques are better able to capture many complex phenomena for which we do not yet know how to make symbolic reasoning effective. Further, the accuracy of the simulators avoids some errors (effectively quantization errors) that abstract, simplified models are subject to.

This method also has some disadvantages. Instead of a single, symbolic simulation to assimilate the process, the method requires a large number of computationally expensive numerical simulations. For this reason, the number of input and output variables that can be studied is limited. Also, this method treats the entire process as a "black box." This leads, as we have seen, to a less densely connected causal graph, because internal points of interaction are not identified. The example we

discussed in Chapter 3 and Chapter 6 illustrates this: the AESOP knowledge base associated under-etching during the polysilicon gate etch step directly with an increased channel length. The intermediate anomaly of an increase in the width of the polysilicon wire forming the gate was not identified. This implies that the hypothesis that under-etching is responsible for an increased channel length cannot be verified by other manifestations of the increased width in the polysilicon wire. Finally, this method is still fairly labour intensive and computationally expensive. The variables of interest must be selected, the experiments designed and run, and the results of the simulations must be manually encoded as causal associations. As an example, the AESOP knowledge base related only 6 processing variables to 6 physical structure variables and 20 measurement variables (each variable assumes four or five values). This required 1100 simulations (1-dimensional SUPREM-III and 2-dimensional PISCES simulations) and 61 CPU hours of compute time on a Convex C1 mini-supercomputer.

## 8.4. Semiconductor Process Representation

This work touches on the problem of serving the needs of all tasks that require a description of the semiconductor manufacturing process with a single, unified representation of it. This is an area of much active research.

Early efforts in this area took a linguistic approach. This approach stresses the similarity between manufacturing plans and software programs, and emphasizes the role of the representation as a set of instructions for carrying out the manufacturing process. The FABLE project [38] used an imperative procedural ALGOL-like language. It identified several levels of description and took macro-expansion as the model for the relationship between these levels. The higher levels of description were concerned with the purpose of each operation. The lower levels successively added detail concerning how to accomplish that purpose. The Berkeley Process Flow Language (BPFL) [40, 41, 102] also takes a linguistic approach. However, their LISP-based language is better able to accomodate the information requirements of multiple tasks.

The programming metaphor is powerful, but also confining. There are many activities requiring a process representation for which the “set of instructions” view is not appropriate. The support of such activities as analysis, real-time process control, scheduling and planning require the inclusion of information that does not readily find a place in such a representation. The primary operations associated with programs are syntax checking, transformation (macro-expansion and compilation) and execution.

Byron Davies [43] did a careful study of the myriad kinds of information that a process representation must include in order to support multiple tasks. Duane Boning developed a general framework for organizing information about semiconductor manufacturing processes [103] . He and George Koppelman have outlined the requirements for a semiconductor process representation [104] .

The major alternative to the linguistic approach is the object-oriented approach [48]. This approach was eventually adopted by the FABLE project at Stanford [39]. The Manufacturing Science Program there has adopted the Distributed Information Service (DIS) [36] as its vehicle for research in this area, and developed the Specification Editor [105, 106] based on it. The Process Flow Representation (PFR) of the MIT Computer - Aided Fabrication Environment (CAFE) [45] also uses this approach. The approach is similar to the AI notion of a framed-based knowledge representation. Computational objects are employed to represent all the objects of interest in the semiconductor world. In addition to process flows, this can include pieces of equipment, personnel, wafers, wafer lots, instances of fabrication runs, test programs, inspection and measurement data, device designs and simulation results. Objects have slots that can contain descriptive information or represent relationships to other objects. They are organized into type hierarchies, and information can be inherited from objects higher in the type hierarchy to objects lower in it.

The efficacy for process representation of the object-oriented approach was demonstrated by Jack Wenstrand [47, 107], who showed that inheritance and reuse of objects significantly reduces the effort needed to specify a manufacturing process, and that the same representation could be used both to generate a runsheet for the factory floor and an input deck for a simulator. Don Basile [51] has investigated the issue of representing the special timing and flow control information associated with semiconductor manufacturing within this framework.

The object-oriented approach, with a client-server architecture and persistent storage for the objects, is the approach being investigated as a possible standard for the semiconductor industry by the CAD Framework Initiative Technology CAD Semiconductor Process Representation Working Group [42, 46, 108], which emphasizes computer-aided design for manufacturing processes, and by the Microelectronics Manufacturing Science Technology (MMST) program [109] and SEMATECH [110], which emphasize computer-integrated manufacturing.

## 8.5. Model Selection

Chapter 5 contains a brief discussion of the issue of how the semiconductor process representation (SPR) required by a particular application can be extracted from the unified SPR. The solution advocated there treats the problem as one of model selection. That is, the problem is reduced to identifying the appropriate application-specific model for each object in the unified SPR. The approach to model selection described there is very flexible, allowing for the possibility that different applications can handle this problem in different ways. However, the primary method suggested was to rely on the abstract classifications of the step to indicate the appropriate model.

As mentioned in Chapter 5, the MIT CAFE [45] system adopts a different approach. It attempts to identify the correct way to model each step by looking only at the step's parameters. Our model selection approach is sufficiently flexible that both methods can be used. The same application can even employ different methods in different parts of the process. In particular, the MIT approach can be used to resolve ambiguous model assignments in the step classification approach.

Both these approaches assume that there is an appropriate model that can be identified before the application begins to execute the model. This may not always be the case. It may not be possible to correctly identify the most appropriate model until some reasoning (or computation) has been performed. Sanjaya Addanki, Roberto Cremonini and Scott Penberthy [111] have addressed this issue. They introduced the notion of a "graph of models." In their scheme, an initial model is identified and applied to the problem-solving task. During this application, it can become clear that the model is inadequate to solve the problem. An *external conflict* occurs when the model's predictions disagree with empirical evidence. Another way for inadequacy of the model to manifest itself is that the model generates contradictory conclusions. This is called an *interal conflict*.

For example, in our situation the model is a description of the how the effects of a processing step are determined by the state of the wafer and the processing conditions. Generally, the model is given descriptions of both the initial wafer state and processing conditions, and the expected outcome of processing (in the form of constraints attached to the step). If the model is not adequate, it may conclude that the effects of the processing step are different from those expected. The validity of a model is predicated on the truth of modeling assumptions associated with the model. Another way for the

application of the model to generate a contradiction is for it to conclude that the situation violates a modeling assumption. For example, a model of an oxidation step might assume that the oxide thickness never reaches the stage where the process is transport limited, and so a linear model of the growth rate is acceptable. However, using that linear model the system might predict that the oxide reaches a thickness for which the oxidation process would be transport limited.

When the model proves inadequate to the task, the system should seek to identify a more appropriate model. In the graph-of-models approach, the models are related to each other in a graph. The models are adjacent (connected to each other) in the graph if they are alternate models for similar situations. The arcs of the graph are annotated with information that helps to decide which alternate model to try next based on the nature of the failure. In particular, the arcs are labelled with the modeling assumptions that have to be changed in order to move from one model to the other. In the graph-of-models methodology, the simplest (having the greatest number of simplifying assumptions) apparently appropriate model is attempted first. Based on the nature of the failure of this model to solve the problem, assumptions are relaxed and increasingly more complex models are then attempted in succession until one is found that proves adequate.

The graph-of-models approach still requires that there are models for every situation explicitly represented before the situation to reasoned about is presented. As we saw in the discussion of diffusion models in Chapter 4, for some domains combinatorics in the characteristics of the problems can make the space of possible models extremely large: sufficiently large to make an explicit representation of all models impractical. Brian Falkenhainer and Ken Forbus [112] and Pandu Nayak [113] and have investigated methods by which the most appropriate model is assembled from model fragments. In this way the space of possible models is represented implicitly by all the possible ways that model fragments can be composed into models.

The key notions are that it is possible to define model fragments that can be combined to form models, modeling assumptions and constraints can be associated with these fragments and features of the problem can be used to direct the search for an appropriate combination of model fragments. The model fragments are organized into equivalence classes based whether they make contradictory assumptions (if two model fragments make inconsistent assumptions then only one of them can be included in a

model at any given time). The model fragments are partial descriptions of phenomena. A relation over them indicates for each model fragment what classes of model fragments can be used to complete them. The features of the problem that constrain the selection of model fragments include constraints from the structure of the system and the behaviour of the system as determined by an initial model. In the work of Falkenhainer and Forbus, the problem to be solved takes the form of a query from the user and the terms employed in the query provide additional constraints on the adequacy of a model. In the work of Nayak, the problem to be solved is always to provide a causal explanation of the system's behaviour. Thus, the expected behaviour provides additional motivation for and constraints on the inclusion of model fragments.

The domain of application for model fragments is physical systems describable by sets of algebraic or differential equations in real variables. The model fragments are themselves sets of equations that partially describe a phenomenon.

Our discrete action models of semiconductor manufacturing operations can be composed sequentially to form a model of the manufacturing plan. Also, the individual models are partly determined by composition in that the parameters, effects and constraints of a model are formed by a cumulative inheritance mechanism. The set of sentences that comprise the effects description of a discrete action model include those local to the model as well as those of all its ancestors in the hierarchy. However, it is not possible to combine two or more action models, e.g. a diffusion model and an oxidation model, to obtain a single model in which the effects of all the models operate concurrently (e.g. a model of oxidation enhanced diffusion).

To achieve compositional modeling of discrete actions, one must combine declarative sentences concerning topological changes to wafer structure. This is considerably more difficult than combining equations. With equations, model fragments interact only through shared variables. Model fragments describing phenomena that would interact in a more complex fashion are excluded from being composed by virtue of the fact that they make contradictory assumptions — each assumes that the interfering phenomenon from the other model is not occurring. It is necessary to have a separate model fragment that encapsulates the interaction between the phenomena.

Given that model fragments in the discrete action world would have sentences referring to the creation and destruction of individuals, and thus to changes in the

topological arrangements among individuals, it is an interesting question as to how any two model fragments affecting overlapping sets of individuals could be written so as not to assume that the other model cannot be operating simultaneously. While it is not obvious how this could be achieved, neither is it obvious that it cannot be.

The work of Janet Murdock [114] goes one step further. In all prior work on model-based reasoning, including the work on compositional modeling, the set of individuals whose behaviour is to be modeled is determined a priori and fixed. The system to be reasoned about is decomposed into a set of interconnected and interacting discrete individuals. Murdock points out that in many cases the underlying physical system is continuous and the most appropriate way to perform this decomposition depends on the nature of the problem being solved. This can include the nature of the question being asked and the behavioural state of the system. For example, a pipe in a chemical plant might be considered an atomic individual in some circumstances, and a composition of several connected sub-pipes in another. Murdock provides a mechanism for performing this individuation automatically as an integral part of model-building. This requires that the reasoning system maintain a representation of the continuous physical geometry of the system being reasoned about that it can “carve up” into individuals as needed to support its reasoning task.

Our discrete action models of semiconductor manufacturing operations also individuate on the fly — an initially homogenous wafer is sub-divided laterally by photolithography operations and vertically by implantation operations. However, they do not entertain different views of the individuation as needed. Once a individual is sub-divided, it remains sub-divided even when it would make more sense to consider it as an undivided whole. For example, when a “blanket layer” is deposited on the wafer, it makes sense to consider the wafer and the deposited layer as single individuals, regardless of whether the wafer has been sub-divided by previous photolithography operations. Instead, the discrete action system considers the deposited layer to be composed of a set of layer-regions, each corresponding to a distinct region-type on the wafer. Furthermore, to simplify the representation of geometry, the discrete action models sub-divide layer-regions unnecessarily. Photolithography operations cause all layers to be sub-divided, in anticipation of the possibility that because the photoresist at the surface of the wafer is differentiated, all layers below it may eventually see differential processing.

# Chapter 9. Conclusions

This chapter discusses some potential avenues of future research. It then briefly summarizes the dissertation and reiterates the contributions of the research.

## 9.1. Future Work

We have focussed on the form and content of the knowledge required to perform diagnosis of semiconductor manufacturing processes. We have omitted discussion of diagnostic strategies. Diagnostic strategy is a non-trivial research topic to which many have applied their minds [5, 53]. An obvious direction in which to extend this work is to incorporate the ideas that have been developed in the area of diagnostic strategies, and to investigate whether this problem domain requires the development of new strategies.

The experiential causal associations include a qualitative, subjective measure of likelihood or certainty. The theoretical models implicitly assume the world is deterministic. There is a need to investigate how statistical information and uncertain reasoning can be incorporated into the diagnosis. Statistical information regarding the frequency of different classes of failures can be used to help order the diagnostic hypotheses. Also, statistical and information theoretic concepts can be used to assess the reliability of the data on which diagnoses are to be based [115, 116]. It would also be interesting to consider whether symbolic models can be extended to include stochastic as well as deterministic notions of causality.

The discussion of the diffusion model in Chapter 4 points up the need to discover ways to achieve compositionality in discrete action models. When describing qualitative criteria for topological changes to wafer structure within the discrete action paradigm, are there conceptualizations that would allow for a superposition principle? The bond-graph and Qualitative Process Theory paradigms are already capable of compositional modeling. Is there a useful level of abstraction at which these techniques can apply in this domain? How can the techniques for qualitative modeling be extended to treat systems characterized by partial differential equations?

It is probably unreasonable to expect that symbolic reasoning techniques will be developed for the systems characterized by partial differential equations. AI techniques generally attempt to achieve the competence demonstrated by human reasoners, and usually lag far behind them. There is little evidence that people can reason qualitatively about such systems. It is interesting to note how people do handle them, and how this compares with the DAS simulator.

The DAS simulator is not often employed as a predictor of behaviour. Instead, it is given information about the outcome of each action, and it builds a causal explanation for the expected outcome. This, I believe, is how people tend to operate when faced with complex systems. They employ numerical techniques and experiments to form predictions, then use their conceptual understanding to interpret the results. Their qualitative conceptual knowledge is not used to predict, but rather to rationalize.

This suggests that a better way to perform symbolic simulation is not to simulate symbolically at all. Rather, the system should be simulated quantitatively, and the symbolic models used to generate a post-hoc rational reconstruction of the result.

In the chapter on related work, we briefly discussed Janet Murdock's work on context-dependent individuation. Context-dependent individuation demands that there be a "basic" representation that makes a minimal number of ontological commitments concerning individuation, and that accurately describes the three-dimensional geometry of the system. When employing symbolic reasoning as a rationalizer/explainer for numerically computed predictions, the fine-grained gridded representation used by the numerical simulator could serve as the basic representation. The symbolic "rationalizer" would then be free to choose the manner in which that representation should be quantized and individuated on the basis of the needs of the models being used to construct the explanation.

One difficulty with this approach is that it subsumes a version of the visual perception problem. The system must be able to use its models to perceive a huge array of numbers as a configuration of larger individuals. However, the quantitative representations of wafer structure are not as undifferentiated as the image data in computer vision: the points in the representation are already identified as to material, for example. Also, unlike the visual perception problem, there is no loss of information due to projection of the three-dimensional world into two dimensional images.

## 9.2. Summary and Contributions

This dissertation is about representing knowledge concerning semiconductor manufacturing that can support symbolic end-to-end diagnosis of stable semiconductor processes in a manner that enables automatic adaptation of the knowledge base to new processes and test structures.

Previous work in this domain has relied on experiential knowledge captured in hand-crafted heuristic rules. The main drawback of this approach has been that such knowledge is not amenable to automatic adaptation to new contexts. Thus, the principle concern of this work has been to address this drawback: to represent the knowledge in a fashion that is robust in the face of changes to the manufacturing process and/or to the structures used to characterize and test the success of the process.

Our solution to this problem has two principle aspects :

1) We represented theoretical knowledge concerning semiconductor manufacturing in the form of declarative, causal models. These models are capable of assimilating a description of the process to be diagnosed and descriptions of the devices and test-structures in the form of the masks used to create them, and creating from these a representation of the causal pathways of the process that can support diagnostic reasoning.
2) We represented experiential diagnostic knowledge concerning phenomena that lie outside the realm of the theoretical knowledge captured in our models. To enable adaptation of this knowledge to new processes and test-structures, we developed a new representation, called the Generic Rule, to integrate experiential knowledge and theoretical knowledge. Generic Rules use theoretical knowledge to describe how the inferences sanctioned by the experiential knowledge depend on the manufacturing plan and the devices and test-structures.

Semiconductor manufacturing processes are extremely complex in two different senses of the word. One sense of the word refers to the complexity of each individual manufacturing operation. This aspect of complexity makes it difficult to model the manufacturing operations accurately, or to define abstract levels of description that do not discard too much information. The other sense of the word refers to the length of the manufacturing plan, the large number of variables that characterize it, and the multiplicity of ways that operations which are temporally remote can interact. We have focussed on dealing with the latter form of complexity, taking advantage of the proficiency with which computers can manage large amounts of data and systematically track large numbers of relationships among them.

Our focus on the latter form of complexity enables us to employ highly abstract models of the manufacturing operations. Indeed, a high level of abstraction is necessary to make diagnostic reasoning tractable. However, the fact of the first form of complexity means there is always a strong tension between abstraction on the one hand and accuracy and generality on the other. The ontological commitments one makes in the name of simplification for tractability of reasoning severely limit the scope of phenomena that can be adequately captured in a general way. We have tried to walk the tightrope between oversimplification and excessive complexity, defining symbolic, qualitative discrete action models of the semiconductor manufacturing operations that most affect the geometry and topology of the structures produced on the wafer.

Experiential knowledge is usually represented at even higher levels abstraction and granularity, in modular heuristic rules. Yet these representations can capture a wider range of phenomena. The cost engendered by this approach is a lack of generality, and a concommitent lack of robustness.

To simultaneously maximize the range of phenomena that can be represented (what we call Phenomena Scope) and maximize the range of contexts for which the knowledge is valid (what we call Situation Scope), we combine the features of model-based representation of general theories with rule-based representation of experiential knowledge in a synergistic manner. The high level of abstraction and coarse granularity that afford flexibility to heuristic rules is retained, augmented by representations based on theoretical knowledge, at a lower level of abstraction and a finer granularity, of the criteria determining when and among which individuals the inference sanctioned by the rules are valid in new contexts.

We have performed several experiments to evaluate our approach. We have created 15 generic rules to generalize the causal associations contained in the AESOP knowledge base regarding the Stanford CMOS process. A simplified version of this process was qualitatively simulated, and the resulting causal dependency network was used to find instances of the generic rules. In this way we assured ourselves that the rules captured the knowledge in the original AESOP knowledge base. All but one of the AESOP causal associations were recovered in this way, and about twenty additional associations were found. Furthermore, some of the causal associations were represented by chains of up to three associations, thereby making explicit the intermediate states involved in the phenomena.

We also successfully extracted a representation of the Stanford BiCMOS process from the Distributed Information Server, and qualitatively simulated that process using our theoretical models. The resulting dependency network was employed both to find instances of the generic rules, and to perform several diagnoses. These diagnoses extended to the process level the diagnoses performed by Greg Freeman's MERLIN system.

Over 500 instances of the generic rules pertaining to the BiCMOS process were obtained. Of these, the validity of roughly one third of the associations is questionable. Some of the associations are invalid because the generic rules ignore sensitivity of the functional dependency involved. In some cases this sensitivity may be so small as to make the effect negligible. A larger number are invalid because of an incorrect assessment of the mechanism responsible for the phenomena that gave rise to the original associations. Despite these shortcomings, we believe that we have demonstrated that the techniques can be successfully employed to improve the robustness of the representations of experiential knowledge.

There is no sign on the horizon that the rapid pace of development in the semiconductor industry will abate, and the trend is inexorably toward ever longer and more complex manufacturing processes. Improvements in the ability to measure and control the effects of individual processing operations will reduce the need for diagnosis at the whole-process level, and may enable even more abstract representations to suffice for diagnosis at that level. However, there will always be a need to keep track of the complex ways that manufacturing operations can interact. As manufacturing processes get longer it will become more important to do this in a systematic way, employing computer tools such as are represented by this work and future extensions of it.

# References


1. Mohammed, J.L. and P. Losleben, *Applications of Artificial Intelligence to Semiconductor Modeling.* (Technical Report No. KSL-93-63). 1993. Knowledge Systems Laboratory, Stanford University.

2. Bobrow, D.G., ed. *Qualitative Reasoning About Physical Systems*. 1985, Cambridge, MA: MIT Press.

3. Davis, R., *Diagnostic reasoning based on structure and behavior.* Artificial Intelligence, 1984. **24**: p. 347–410.

4. Freeman, G.G. and W. Lukaszek, *MERLIN: an automated IC process diagnosis tool based on analytic device models,* in *The Stanford BiCMOS Project,* J.D. Shott, Editor. 1990, Stanford University. p. 197–220.

5. Freeman, G.G., *Application of Analytic Models to Computer-aided IC Device Diagnosis and Parametric Test Selection.* Ph.D. thesis, 1991. Stanford University.

6. Mohammed, J.L. and R.G. Simmons, *Qualitative Simulation of Semiconductor Fabrication*, in *AAAI-86*. Held: Philadelphia, PA, August 11–15, 1986. 1986, USA: Morgan Kaufman. p. 794–9.

7. Simmons, R.G. and J.L. Mohammed, *Causal Modeling of Semiconductor Fabrication.* Artificial Intelligence in Engineering, 1989. **4**(1): p. 2–21.

8. Mohammed, J.L. and P. Losleben, *Process Representation and Model Selection.* (Manufacturing Science Technical Note Series, Paper No. 2). 1992. Stanford Integrated Circuits Laboratory.

9. Mohammed, J.L. and P. Losleben, *Representation of Mask Information in the MKS Process Representation.* (Manufacturing Science Technical Note Series, Paper No. 1). 1992. Stanford Integrated Circuits Laboratory.

10. Pan, J.Y.-C. and J.M. Tenenbaum, *PIES: An engineer's do-it-yourself knowledge system for interpretation of parametric test data.* AI Magazine, 1986. **7**(4): p. 62–9.

11. Dishaw, J.P. and J.Y.-C. Pan, *AESOP: a simulation-based knowledge system for CMOS process diagnosis.* IEEE Transactions on Semiconductor Manufacturing, 1989. **2**(3): p. 94–103.

12. Hayes-Roth, F., D.A. Waterman, and D.B. Lenat, *Building Expert Systems.* Teknowledge series in knowledge engineering, 1983, Reading, MA: Addison-Wesley Publishing Company. 444.

13. Weiss, S.M., C.A. Kulikowski, and A. Safir, *A model-based consultation system for the long-term management of glaucoma.*, in *International Joint Conference on Artificial Intelligence*. Held: Cambridge, MA, 1977, USA: Morgan Kaufman. p. 826–832.

14. Simmons, R.G., *Representing and Reasoning About Change in Geologic Interpretation.* (AI Technical Report No. 749). 1983. MIT.

15. Iwasaki, Y. and H.A. Simon, *Causality and model abstraction.* Artificial Intelligence, 1994. **67**(1): p. 143–94.

16. Patil, R.S., P. Szolovits, and W.B. Schwartz, *Causal understanding of patient illness in medical diagnosis.*, in *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*. 1981, International Joint Conferences on Artificial Intelligence, Inc. p. 893–899.

17. de Kleer, J. and J.S. Brown, *A qualitative physics based on confluences.* Artificial Intelligence, 1984. **24**: p. 7–83.

18. Forbus, K.D., *Qualitative process theory.* Artificial Intelligence, 1984. **24**: p. 85–168.

19. Kuipers, B., *Qualitative Simulation.* Artificial Intelligence, 1986. **29**: p. 289–338.

20. Genesereth, M.R., *The use of design descriptions in automated diagnosis,* in *Qualitative Reasoning About Physical Systems,* D.G. Bobrow, Editor. 1985, Cambridge. MA: The MIT Press. p. 411–436.

21. Barrow, H.G., *VERIFY: A program for proving the correctness of digital hardware designs,* in *Qualitative Reasoning About Physical Systems,* D.G. Bobrow, Editor. 1985, Cambridge, MA: The MIT Press. p. 437–492.

22. Mendelson, E., *Introduction to Mathematical Logic*. 1964, Princeton, NJ: D. Van Nostrand. .

23. McCarthy, J. and P.J. Hayes, *Some philosophical problems from the standpoint of Artificial Intelligence.* Machine Intelligence, 1969. **4**: p. 463-502.

24. Shoham, Y., *Chronological Ignorance: Experiments in Nonmonotonic Temporal Reasoning.* Artificial Intelligence, 1988. **36**: p. 279-331.

25. Weyhrauch, R.W., *Prolegomena to a theory of mechanized formal reasoning.* Artificial Intelligence, 1980. **13**: p. 133-170.

26. Roussel, P., *Prolog: Manuel de reference et d'utilisation.* 1975. Groupe d'Intelligence Artificielle, Marseille-Luminy.

27. Warren, D.H.D., *Logic Programming and Compiler Writing.* (Res. Rep No. 44). 1977. Dept. of Artificial Intelligence, University of Edinburgh.

28. Simmons, R.G., *"Commonsense" Arithmetic Reasoning*, in *AAAI-86*. Held: Philadelphia, PA, 1986, Morgan Kaufman. p. 118–124.

29. McAllester, D., The Use of Equality in Deduction and Knowledge Representation. (AI Technical Report No. 550). 1980. Massachusetts Institute of Technology.

30. Chapman, D., *Planning for Conjunctive Goals.* (Artificial Intelligence Technical Report No. 802). 1985. Massachusetts Institute of Technology.

31. Fikes, R.E. and N.J. Nilsson, *STRIPS: A new approach to the application of theorem proving to problem solving.* Artificial Intelligence, 1971. **2**: p. 189-208.

32. Sacerdoti, E.D., *A structure for plans and behavior.* (Technical Note No. 109). 1975. AI Center, SRI International Inc., Menlo Park, CA.

33. Goosens, R.J.G. and R.W. Dutton, *Device CAD in the 90's: At the Crossroads,* in *IEEE Circuits & Devices Magazine*. 1992, . p. 18–26.

34. Law, M.E., C.S. Rafferty, and R.W. Dutton, *SUPREM–IV.* (Technical Report) 1988. Stanford Electronics Laboratories, Stanford University, Stanford, CA.

35. McVittie, J.P., *et al.*, *SPEEDIE: Simulation of Profile Evolution During Etching and Deposition*, in *SPIE Symposium on Advanced Techniques for Integrated Circuits Processing*. 1990, p. 126-138.

36. Glicksman, J., *DIS: The Distributed Information Service.* (Reference and User's Guide, Version No. 2.0). 1992. Enterprise Integration Technologies.

37. Strojwas, A.J. and S.W. Directory, *The process engineer's workbench.* IEEE Journal of Solid State Circuits, 1988. **23**(2): p. 377-386.

38. Ossher, H.L. and B.K. Reid, *FABLE: A programming language solution to IC process automation problems.* ACM-SIGPLAN Notices, 1983. **18**(6): p. 137-148.

39. Davies, B., *et al.*, *FABLE: Knowledge For Semiconductor Manufacturing*. 1987, The Center for Integrated Systems, Stanford University (Unpublished).

40. Williams, C.B. and L.A. Rowe, *The Berkeley Process-Flow Language: Reference Document.* (Report No. UCB/ERL M87/73). 1987. Electronics Research Laboratory, University of California, Berkeley.

41. Williams, C.B., *Design and Implementation of the Berkeley Process-Flow Language Interpreter.* (Master's Thesis No. UCB/ERL M88/72). 1988. Electronics Research Laboratory, University of California, Berkeley.

42. SPR Working Group of the CAD Framework Initiative TCAD TSC, *Semiconductor Process Representation Requirements.* (CFI Document No. TCAD-91-G-4). 1992. CAD Framework Initiative, Inc.

43. Davies, B., *Process Specification and Process Knowledge in Semiconductor Manufacturing.* Ph.D. thesis, 1991. Stanford University.

44. McIlrath, M.B. and D.S. Boning, *Integrating process design and manufacture using a unified flow representation*, in *Second International Conference on Computer Integrated Manufacturing*. Held: Troy, NY, 1990, IEEE Computer Society Press, Los Alamitos, CA. p. 224–230.

45. McIlrath, M.B., *et al.*, *CAFE—the MIT Computer-Aided Fabrication Environment.* IEEE Transactions on Components, Hybrids, and Manufacturing Technology, 1992. **15**(3): p. 353–60.

46. SPR Working Group of the CAD Framework Initiative TCAD TSC, *Current Concepts in Semiconductor Process Representation.* (CFI Document No. TCAD-91-T-2). 1991. CAD Framework Initiative, Inc.

47. Wendstrand, J.S., *An Object-Oriented Model for Specification, Simulation, and Design of Semiconductor Fabrication Processes.* (Thesis No. ICL91-003). 1991. Stanford Electronics Laboratories, Stanford University.

48. Pan, J.Y.-C., J.M. Tenenbaum, and J. Glicksman, *A framework for knowledge-based Computer Integrated Manufacturing.* IEEE Transactions on Semiconductor Manufacturing, 1989. **2**(2): p. 33–46.

49. Schmolze, J.G. and T.A. Lipkis, *Classification in the KL-ONE Knowledge Representation System*, in *International Joint Conference on Artificial Intelligence.* Held: Karlsruhe, Germany, 1983, William Kaufman, Inc., Los Altos, CA. p. 330-332.

50. Shott, J., *Overview of the Stanford BiCMOS Process*. 1988, Integrated Circuits Laboratory, Stanford University (unpublished).

51. Basile, D., *Flow Control for Object-Oriented Semiconductor Process Representations.* (Thesis No. ICL No. 92-018). 1992. Integrated Circuits Laboratory, Stanford University.

52. de Kleer, J., *Local methods for localizing faults in electronic circuits.* (Memo No. 394). 1976. MIT Artificial Intelligence Laboratory.

53. de Kleer, J. and B.C. Williams, *Diagnosing multiple faults.* Artificial Intelligence, 1987. **32**(1): p. 97–130.

54. Hamscher, W.C., *Modeling digital circuits for trouble-shooting.* Artificial Intelligence, 1991. **51**(1–3): p. 223–71.

55. Williams, B.C., *Qualitative analysis of MOS circuits.* Artificial Intelligence, 1984. **24**: p. 281–346.

56. First, M.B., *et al.*, *LOCALIZE: Computer-assisted localization of peripheral nervous system lesions.* Computers and Biomedical Research, 1982. **15**(6): p. 525–543.

57. Uckun, S., B.M. Dawant, and D.P. Lindstrom, *Model-based diagnosis in intensive care monitoring: the YAQ approach.* Artificial Intelligence in Medicine, 1993. **5**(1): p. 31–48.

58. Ursino, M., G. Avanzolini, and P. Barbini, *Qualitative simulation of dynamic physiological models using the KEE environment.* Artificial Intelligence in Medicine, 1992. **4**(1): p. 53–73.

59. Weinberg, J.B., G. Biswas, and L.A. Weinberg, *Adventures in qualitative modeling — a qualitative heart model*, in *IEEE International Conference on Systems, Man and Cybernetics.* Held: Cambridge, MA, USA, 14–17 November 1989. 1989, USA: IEEE. p. 1003–8.

60. Hunt, J.E. and et al, *Applications of qualitative model-based reasoning.* Control Engineering Practice, 1993. **1**(2): p. 253–66.

61. Fruchter, R. and et al, *Qualitative Modeling and analysis of lateral load resistance in frames.* Artificial Intelligence for Engineering Design, Analysis and Manufacturing — AI EDAM, 1993. **7**(4): p. 239–56.

62. Saarela, O., *Model-based control of power plant diagnostics.* International Journal of Electrical Power & Energy Systems, 1993. **15**(2): p. 101–8.

63. Feray-Beaumont, S.E., *et al.*, *Qualitative modelling of distillation columns*, in *Advanced Control of Chemical Processes — ADCHEM*. Held: Toulouse, France, 14–16 October 1991. 1991, UK: Pergammon. p. 191–6.

64. Hangos, K.M., Z. Csaki, and S.B. Jorgensen, *Qualitative model-based intelligent control of a distillation column.* Engineering Applications of Artificial Intelligence, 1992. **5**(5): p. 431–40.

65. Pan, J.Y.-C., *Qualitative reasoning with deep-level mechanism models for diagnoses of mechanism failures*, in *CAIA-84*. Held: Denver, CO, 1984, .

66. Hamilton, T.P., *HELIX: a helicopter diagnostic system based on qualitative physics.* International Journal for Artificial Intelligence in Engineering, 1988. **3**(3): p. 141–50.

67. Keller, R.M., *et al.*, *Compiling Special-Purpose Rules from General-Purpose Device Models.* (Technical Report No. KSL-89-49). 1989. Knowledge Systems Laboratory, Stanford University.

68. Scarl, E., J.R. Jamieson, and C.I. Delaune, *A fault detection and isolation method applied to liquid oxygen loading for the space shuttle*, in *IJCAI-85*. Held: Los Angeles, CA, 1985, San Mateo:Morgan Kaufman. p. 414–416.

69. Hofmann, M.O., T.L. Cost, and M. Whitley, *Model-based diagnosis of the Space Shuttle Main Engine.* Artificial Intelligence for Engineering Design, Analysis and Manufacturing — AI EDAM, 1992. **6**(3): p. 131–48.

70. Hofmann, M.O. and T.L. Cost, *Model-based rocket engine diagnosis*, in *Fourth International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems. IEA/AIE-91*. Held: Kauai, HI, USA, 2–5 June 1991. 1991, USA: Univ. Tennessee Space Inst. p. 266–74.

71. Chen, S.-s., *QUASI: A QUalitative Analysis of Surface and Interface program for microelectronic materials processing*, in *Applications of Artificial Intelligence VIII*. Held: Orlando, FL, USA, April 17–19, 1990. 1990, USA: Proceedings of the SPIE—The International Society for Optical Engineering **1293**, pt. 2. p. 972–83.

72. Hamscher, W., *Model-based reasoning in financial domains.* Knowledge Engineering Review, 1992. **7**(4): p. 323–43.

73. Farley, A.M. and K.-P. Lin, *Qualitative reasoning in microeconomics: an example*, in *Decision Support Systems and Qualitative Reasoning. Proceedings of the IMACS International Workshop*. Held: Toulouse, France, 13–15 March 1991. 1991, Netherlands: North-Holland. p. 303–6.

74. Karakoulas, G.J., *Diagnostic reasoning about an economic system: a model-based approach*, in *Decision Support Systems and Qualitative Reasoning. Proceedings of the IMACS International Workshop.* Held: Toulouse, France, 13–15 March 1991. 1991, Netherlands: North-Holland. p. 293–6.

75. Daniels, H.A.M. and A.J. Feelders, *Combining qualitative and quantitative methods for model-based diagnosis of firms*, in *Decision Support Systems and Qualitative Reasoning. Proceedings of the IMACS International Workshop.* Held: Toulouse, France, 13–15 March 1991. 1991, Netherlands: North-Holland. p. 255–60.

76. Schwartzler, G., *An algebraic approach to knowledge-based modelling*, in *Artificial Intelligence and Symbolic Mathematical Computing. International Conference AISMC-1*. Held: Karlsruhe, Germany, 3–6 Aug. 1992. 1993, Germany: Springer-Verlag. p. 85–95.

77. Crawford, J., A. Farquar, and B. Kuipers, *QPC: A compiler from physical models into qualitative differential equations*, in *AAAI-90*. 1990, San Mateo: Morgan Kaufman Publishers. p. 365–372.

78. Xia, S., D.A. Linkens, and S. Bennett, *Qualitative reasoning and applications to dynamic systems: a bond graph approach*, in *Decision Support Systems and Qualitative Reasoning. Proceedings of the IMACS International Workshop.* Held: Toulouse, France, 13–15 March 1991. 1991, Netherlands: North-Holland. p. 175–80.

79. Williams, B., *Doing time: Putting qualitative reasoning on firmer ground*, in *AAAI-86*. Held: Philadelphia, PA, 1986, San Mateo: Morgan Kaufman Publishers. .

80. Iwasaki, Y. and C.-M. Low, *Model generation and simulation of device behavior with continuous and discrete changes.* (Technical Report No. KSL 91-69). 1991. Stanford University, Knowledge Systems Laboratory.

81. Kuipers, B., *Abstraction by time-scale in qualitative simulation*, in *AAAI-87*. Held: Seattle, WA, 1987, San Mateo: Morgan Kaufman Publishers. p. 621–625.

82. Saxena, S. and A. Unruh, *Diagnosis of Semiconductor Manufacturing Equipment and Processes.* IEEE Transactions on Semiconductor Manufacturing, 1994. 7(2): p. 220–232.

83. Mina, I., *A knowledge-based model for representation and reasoning in manufacturing*, in *Proceedings of the Third International IMS '87 Conference*. Held: Long Beach, CA, USA, 1987, USA: Intertec Commun. p. 270–87.

84. Mina, I., *UMXD: a diagnosts expert system for manufacturing*, in *First Annual ESD/SMI Expert Systems Conference and Exposition for Advanced Manufacturing Technology*. Held: Dearborn, MI, USA, 1987, USA: Eng. Soc. Detroit. p. 281–94.

85. Mina, I., *PCMR: a language for process control modelling and reasoning*, in *Proceedings of the 1987 American Control Conference*. Held: Minneapolis, MN, USA, 10–12 June 1987. 1987, USA: American Autom. Control Council. p. 174–80.

86. Hart, P., *Directions for AI in the Eighties,* in *SIGART Newsletter*. 1982. p. 11–16.

87. Michie, D., *High-Road and Low-Road Programs,* in *AI Magazine*. 1981. p. 21–2.

88. Chandrasekaran, M.S., *Deep vs compiled knowledge approaches to diagnostic problem solving*, in *AAAI 82*. 1982, p. 349–354.

89. Keller, R.M., *et al.*, *Compiling Diagnosis Rules and Redesign Plans from a Structure/Behavior Device Model: The Details.* (Technical Report No. KSL-89-50). 1989. Knowledge Systems Laboratory, Stanford University.

90. Goel, A.K., *et al.*, *Knowledge Compilation: A symposium,* in *IEEE Expert*. 1991. p. 71-92.

91. Davis, R., *Form and Content in Model Based Reasoning*, in *International Joint Conference on Artificial Intelligence*. Held: Detroit, MI, 1989, USA:Morgan Kaufman. p. 11–27.

92. Bylander, T., *A Simple Model of Knowledge Compilation,* in *IEEE Expert*. 1991. p. 73–74, 91–92.

93. Simmons, R.G., *The roles of associational and causal reasoning in problem solving.* Artificial Intelligence, 1992. **53**: p. 159–207.

94. Nassif, S.R., A.J. Strojwas, and S.W. Director, *FABRICS II: a statistically-based IC Fabrication process simulator.* IEEE Transactions on Computer-Aided Design, 1984. **CAD-3** (1): p. 40-46.

95. Slaughter, S., R. Hartzell, and T.-C. Chang, *A deep level reasoning tool for semiconductor process simulation*, in *IEEE/SEMI International Semiconductor Manufacturing Science Symposium—ISMSS '89*. Held: Burlingame, CA, USA, May 22–24, 1989. 1989, USA: IEEE (Cat. No.89CH2699-7). p. 117.

96. Funakoshi, K. and K. Mizuno, *A Rule-Based VLSI Process Flow Validation System With Macroscopic Process Simulation.* IEEE Transactions on Semiconductor Manufacturing, 1990. **3**(4): p. 239–246.

97. Ho, C.P., S.E. Hansen, and P.M. Fahey, *SUPREM III—Program for Integrated Circuit Process Modeling and Simulation.* (Report No. SEL84-001). 1984. Stanford Electronics Laboratories, Stanford University, Stanford, CA.

98. Mulvaney, B.J. and W.B. Richardson, *PEPPER—a process simulator for VLSI.* IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 1989. **8**(4): p. 336–49.

99. Pelka, J., *Simulation of ion-enhanced dry-etch processes.* Proceedings of the SPIE — The International Society for Optical Engineering, 1991. **1392**: p. 55–66.

100. Richardson, W.B. and B.J. Mulvaney, *Modeling phosphorus diffusion in three dimensions.* IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 1992. **11**(4): p. 487–96.

101. Scheckler, E.W. and A.R. Neureuther, *Models and algorithms for three-dimensional topography simulation with SAMPLE-3D.* IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 1994. **13**(2): p. 219–30.

102. Hegarty, C.J., *Process-Flow Specification and Dynamic Run Modification for Semiconductor Manufacturing.* (Thesis No. UCB/ERL M91/40). 1991. Electronics Research Laboratory, University of California, Berkeley.

103. Boning, D.S., *et al.*, *A general semiconductor press modeling framework.* IEEE Transactions on Semiconductor Manufacturing, 1992. **5**(4): p. 266–80.

104. Boning, D.S. and G. Koppelman, *Semiconductor Process Representation Requirements.* (Draft Internal Document No. tcad-911-G-4). 1991. TCAD Framework Group, Semiconductor Process Representation Working Group, CAD Framework Initiative, Inc.

105. Pelts, G.L. and D.G. Basile, *SPEC SYSTEM: Modules User's Guide.* (Technical Note No. 3). 1993. Integrated Circuits Laboratory, Stanford University.

106. Pelts, G.L. and D.G. Basile, *SPEC SYSTEM: Overview — The Launcher.* (Technical Note No. 4). 1993. Integrated Circuits Laboratory, Stanford University.

107. Wendstrand, J.S., H. Iwai, and R.W. Dutton, *A manufacturing-oriented environment for synthesis of fabrication processes.*, in *ICCAD '89*. Held: San Francisco, 1989, IEEE. p. 376-379.

108. Boning, D. and J. Glicksman, *Semiconductor Process Representation Procedural Interface (PI).* (Draft Specification, Version No. 0.7). 1993. CAD Framework Initiative, Inc.

109. McGehee, J., *The MMST Computer-Integrated Manufacturing System Framework.* IEEE Transactions on Semiconductor Manufacturing, 1994. **7**(2): p. 107–16.

110. SEMATECH, *Collaborative Manufacturing System Computer Integrated Manufacturing Application Framework Specification 1.0.* (technology transfer report No. 9306197C-ENG). 1994. SEMATECH.

111. Addanki, S., R. Cremonini, and J.S. Penberthy, *Graphs of Models.* Artificial Intelligence, 1991. **51**: p. 145–77.

112. Falkenhainer, B. and K.D. Forbus, *Compositional modeling: Finding the right model for the job.* Artificial Intelligence, 1991. **51**: p. 95–143.

113. Nayak, P.P., *Automated Modeling of Physical Systems.* (Ph.D. Thesis — Report No. STAN-CS-92-1443, also KSL-92-69). 1992. Department of Computer Science, Stanford University.

114. Murdock, J.L., *Model Matching and Individuation for Model-Based Diagnosis.* 1994. Ph.D. Thesis — in preparation. Department of Computer Science, Stanford University.

115. Chang, N.H. and C.J. Spanos, *Chronological equipment diagnosis using evidence integration*, in *Applications of Artificial Intelligence VIII*. Held: Orlando, FL, USA, April 17–19, 1990. 1990, USA: Proceedings of the SPIE—The International Society for Optical Engineering **1293**, pt. 2. p. 944–55.

116. Shafer, G., *A Mathematical Theory of Evidence*. 1976. Princeton, N.J.: Princeton University Press.

# Appendix A: Modeling Objects

This appendix exhibits the definitions of the classes of temporal objects (and special constants) employed in the semiconductor manufacturing ontology.

```
(defGDobject WAFER (temporal-object)
  ((structures   (set . 2D-structure))
   (layers       (set . layer))
   (region-types (set . region-type))))

;;; Abstract class of objects that keep track of
;;; lateral dimensions and topology
(defGDobject HORIZONTAL-DESCR (temporal-object)
  ((horizontal-regions (set . horizontal-region))))

;;; Class of objects that describe lateral topology of
;;; slices through wafer structures
(defGDobject 2D-STRUCTURE (horizontal-descr)
  ((horizontal-regions (set . wafer-region))
   (left-most wafer-region)))

;;; Class of objects that describe the lateral topology of
;;; masks for a 2D slice through a structure
(defGDobject MASK (horizontal-descr)
  ((horizontal-regions (set . mask-region))
   (left-most mask-region)))

(defGDobject REGION-OR-EDGE (temporal-object))

(defGDobject EDGE (region-or-edge)
  ((left region-or-edge)
   (left-pos finite-real)
   (right region-or-edge)
   (right-pos finite-real)
   (left-wafer-region wafer-region)
   (right-wafer-region wafer-region)
   (region-type region-type)
   (opacity (one-of (constant opaque)
                    (constant transparent)))))
```

```
;;; Class of objects that describe the lateral topology
;;; and dimensions for a single laterally homogeneous region
(defGDobject HORIZONTAL-REGION (region-or-edge)
  ((left region-or-edge)
   (left-pos finite-real)
   (right region-or-edge)
   (right-pos finite-real)))

;;; Class of objects that describe the lateral topology
;;; and dimensions for a single laterally homogeneous region
;;; of a 2D slice through a structure on a wafer
(defGDobject WAFER-REGION (horizontal-region)
  ((region-type region-type)
   (pieces (set . wafer-region))))

;;; Class of objects that describe the lateral topology
;;; and dimensions for a single laterally homogeneous region
;;; of a mask for a 2D slice through a structure
(defGDobject MASK-REGION (horizontal-region)
  ((left-wafer-region wafer-region)
   (right-wafer-region wafer-region)
   (opacity (one-of (constant opaque)
                    (constant transparent)))))

;;; Class of objects that describe opaque 2D mask regions
(defGDobject OPAQUE-MASK-REGION (mask-region)
  ((opacity (constant opaque))))

;;; Class of objects that describe transparent 2D mask regions
(defGDobject TRANSPARENT-MASK-REGION (mask-region)
  ((opacity (constant transparent))))

;;; Class of objects that describe the vertical sequence of
;;; layers in a given region-type
(defGDobject REGION-TYPE (temporal-object)
  ((surface layer-region)
   (regions (set . wafer-region))))

;;; Class of objects that describe a given layer in a given region,
;;; encompassing also the "environment" at the wafer's surfaces
(defGDobject LAYER-OR-ENV (temporal-object))

;;; Class containing one instance: the environment at the wafer's
;;; surface (Same object is used at both front and back wafer surfaces.)
(defGDobject ENVIRONMENT (layer-or-env)
  ((gases (set . gas))))

;;; Class of objects representing layer-regions that have a common genesis
(defGDobject LAYER (temporal-object)
  ((layer-regions (set . layer-region))))
```

*;;; Class of objects representing individual layers in individual region-types*

```
(defGDobject LAYER-REGION (layer-or-env)
  ((region-type region-type)
   (layer layer)
   (top-pos finite-real)
   (above layer-or-env)
   (bottom-pos finite-real)
   (below layer-or-env)
   (material material-type)
   (dopant dopant)
   (concentration finite-non-negative-real)
   (thickness finite-non-negative-real)))

(defGDobject MATERIAL-TYPE (temporal-object))

(defGDobject METAL (material-type))

(defGDobject ALUMINUM (metal))

(defGDobject TUNGSTEN (metal))

(defGDobject SILICON (material-type))

(defGDobject POLYSILICON (silicon))

(defGDobject OXIDE (material-type))

(defGDobject NITRIDE (material-type))

(defGDobject PHOTORESIST (material-type))

(defGDobject EXPOSED-PHOTORESIST (photoresist))

(defGDobject POS-PHOTORESIST (photoresist))

(defGDobject NEG-PHOTORESIST (photoresist))

(defGDobject EXPOSED-POS-PHOTORESIST (exposed-photoresist))

(defGDobject EXPOSED-NEG-PHOTORESIST (exposed-photoresist))

(defGDobject DOPANT (temporal-object)
  ((dopant-type (one-of (constant N) (constant P)))))

(defGDobject N-DOPANT (dopant)
  ((dopant-type (constant N))))
```

```
(defGDobject P-DOPANT (dopant)
  ((dopant-type (constant P))))


(defGDobject ETCHANT-TYPE (temporal-object))


(RUP:INIT-FORM (make-objectq START 'time))
(RUP:INIT-FORM (assert-expr '(< START *NOW*)))
(RUP:INIT-FORM (make-objectq THE-ENVIRONMENT 'environment
                 :existence :always))
(RUP:INIT-FORM
 (make-objectq PHOTORESIST-THICKNESS 'finite-positive-real))
(RUP:INIT-FORM (make-objectq POSITIVE-PHOTORESIST
                 'pos-photoresist :existence :always))
(RUP:INIT-FORM (make-objectq RESIST-1813 'pos-photoresist
                 :existence :always))
(RUP:INIT-FORM (make-objectq NEGATIVE-PHOTORESIST
                 'neg-photoresist :existence :always))
(RUP:INIT-FORM (make-objectq POS-PHOTORESIST-EXPOSED
                 'exposed-pos-photoresist))
(RUP:INIT-FORM (make-objectq NEG-PHOTORESIST-EXPOSED
                 'exposed-neg-photoresist))
(RUP:INIT-FORM (make-objectq PHOTORESIST-EXPOSED
                 'exposed-photoresist))
(RUP:INIT-FORM (make-objectq MASK-EDGE   'edge :existence
                 :always))
(RUP:INIT-FORM (make-objectq REGION-EDGE 'edge :existence
                 :always))
(RUP:INIT-FORM (make-objectq *NO-DOPANT* 'dopant :existence
                 :always))
(RUP:INIT-FORM (make-objectq |Oxide|   'oxide    :existence
                 :always))
(RUP:INIT-FORM (make-objectq |Silicon| 'silicon  :existence
                 :always))
(RUP:INIT-FORM (make-objectq |Polysilicon| 'polysilicon
                 :existence :always))
(RUP:INIT-FORM (make-objectq |Nitride|  'nitride  :existence
                 :always))
(RUP:INIT-FORM (make-objectq |Aluminum| 'aluminum :existence
                 :always))
(RUP:INIT-FORM (make-objectq |Tungsten| 'tungsten :existence
                 :always))
(RUP:INIT-FORM (make-objectq ARSENIC    'n-dopant :existence
                 :always))
(RUP:INIT-FORM (make-objectq PHOSPHORUS 'n-dopant :existence
                 :always))
(RUP:INIT-FORM (make-objectq ANTIMONY   'n-dopant :existence
                 :always))
(RUP:INIT-FORM (make-objectq BORON      'p-dopant :existence
                 :always))
(RUP:INIT-FORM (make-objectq GALLIUM    'p-dopant :existence
                 :always))
(RUP:INIT-FORM
  (history::tforcefully ^(@ (dopant-type ARSENIC) *NOW*)))
(RUP:INIT-FORM
  (history::tforcefully ^(@ (dopant-type PHOSPHORUS) *NOW*)))
(RUP:INIT-FORM
  (history::tforcefully ^(@ (dopant-type ANTIMONY) *NOW*)))
```

```
(RUP:INIT-FORM
  (history::tforcefully ^(@ (dopant-type BORON) *NOW*)))
(RUP:INIT-FORM
  (history::tforcefully ^(@ (dopant-type GALLIUM) *NOW*)))
(RUP:INIT-FORM (make-objectq |6:1 Buffered HF| 'etchant-type
                 :existence :always))
(RUP:INIT-FORM (make-objectq REFLUXED-H3PO4    'etchant-type
                 :existence :always))
(RUP:INIT-FORM (make-objectq |50:1 HF|         'etchant-type
                 :existence :always))
(RUP:INIT-FORM
 (assert-expr '(N= Etch-Angle-Threshold 85.0) 'DOMAIN-MODEL))
(RUP:INIT-FORM
 (assert-expr '(>  Poly-Etch-Etch-Rate   0.0) 'DOMAIN-MODEL))
```

# Appendix B: Modeling Actions

This appendix exhibits the discrete action models of the semiconductor manufacturing operations.

*;;; Abstract parent of all processes that add a new “blanket” layer at the wafer surface.*

```
(defprocess DAS-ADD-TOP-LAYER
  :NON-EXECUTABLE
  parameters (($mat material-type))
  effects
  ((for-all-existing $ld \: Region-Type (start-of (process interval))
     (:=>
      (pred1 (old-top $ld (process interval))
             (start-of (process interval)))
      (CREATED ($NL layer-region (start-of (process interval)))
       (member $NL (new-layer $mat (process interval)))
       (change = (surface $ld) $nl)
       (change = (region-type $nl) $ld)
       (change = (top-pos $nl)
        (+ (mat-thick $ld (process interval))
           (old-top-pos $ld (process interval))))
       (change = (above $nl) THE-ENVIRONMENT)
       (change = (bottom-pos $nl) (old-top-pos $ld (process interval)))
       (change = (below $nl) (old-top $ld (process interval)))
       (change = (material $nl) $mat)
       (change = (thickness $nl) (mat-thick $ld (process interval)))
       (change = (above (old-top $ld (process interval))) $nl))))
   (CREATED ($New-Layer layer (start-of (process interval)))
     (for-all-true $l e (new-layer $mat (process interval))
       (:AND (member $l (@ (layer-regions $new-layer)
                           (end-of (process interval))))
             (change = (layer $l) $new-layer))) ))
  constraints
  ((DEFN OLD-TOP-POS (ld i) (@ (top-pos (surface ld)) (start-of i)))
   (DEFN OLD-TOP (ld i) (@ (surface ld) (start-of i)))))


(defprocess DAS-SPIN-ON-RESIST
  AKO DAS-ADD-TOP-LAYER
  parameters (($mat photoresist))
  constraints
  ((DEFN PRED1 ($lr $t)
    (adherent $mat (@ (material $lr) $t)))
   (DEFN MAT-THICK ($ld $i)
    (- PHOTORESIST-TOP-POS (OLD-TOP-POS $ld $I)))))
```

```
(defprocess DAS-SPUTTERING
  AKO DAS-add-top-layer
  parameters (($mat metal) ($duration finite-positive-real))
  constraints
  ((n= (duration (process interval)) $duration)
   (DEFN MAT-THICK ($ld $i) (SPUTTERING-THICKNESS $duration))
   (> (SPUTTERING-THICKNESS $duration) 0.0)
   (DEFN PRED1 ($lr $t)
    (adherent $mat (@ (material $lr) $t)))))
(init-form
 (assert-expr '(FUNCTIONAL SPUTTERING-THICKNESS MI)))


(defprocess DAS-CHEMICAL-VAPOR-DEPOSITION
  :NON-EXECUTABLE
  AKO DAS-add-top-layer
  parameters
  (($mat material-type) ($duration finite-positive-real)
   ($temperature finite-positive-real)
   ($vapor-pressure finite-positive-real))
  constraints
  ((n= (duration (process interval)) $duration)
   (DEFN MAT-THICK ($ld $i)
    (CVD-THICKNESS $duration $temperature $vapor-pressure))
   (> (CVD-THICKNESS $duration $temperature $vapor-pressure)
      0.0)
   (DEFN PRED1 ($lr $t)
    (adherent $mat (@ (material $lr) $t)))))
(init-form
 (assert-expr '(FUNCTIONAL CVD-THICKNESS MI MI MI)))


(defprocess DAS-UNDOPED-CHEMICAL-VAPOR-DEPOSITION
  AKO DAS-CHEMICAL-VAPOR-DEPOSITION
  parameters ()
  effects
  ((for-all-existing $ld \: Region-type
                     (end-of (process interval))
    (change = (dopant (@ (surface $ld)
                         (end-of (process interval))))
      *no-dopant*))))


(defprocess DAS-DOPED-CHEMICAL-VAPOR-DEPOSITION
  AKO DAS-CHEMICAL-VAPOR-DEPOSITION
  parameters
  (($dopant dopant) ($concentration finite-positive-real))
  effects
  ((for-all-existing $ld \: Region-type
                     (end-of (process interval))
    (:AND
     (change = (dopant (@ (surface $ld)
                       (end-of (process interval))))
       $dopant)
     (change =
       (concentration (@ (surface $ld)
                         (end-of (process interval))))
       $concentration)))))
```

```
(defprocess DAS-EPITAXIAL-GROWTH
  AKO DAS-add-top-layer
  parameters
  (($mat silicon)
   ($duration finite-positive-real)
   ($temperature finite-positive-real)
   ($dopant dopant) ($concentration finite-positive-real))
  effects
  ((for-all-existing $ld \: Region-type
                     (end-of (process interval))
    (:AND
     (change = (dopant (@ (surface $ld)
                          (end-of (process interval))))
       $dopant)
     (change =
      (concentration (@ (surface $ld)
                        (end-of (process interval))))
      $concentration))))
  constraints
  ((n= (duration (process interval)) $duration)
   (DEFN MAT-THICK ($ld $i)
    (EPI-THICKNESS $duration $temperature))
   (> (EPI-THICKNESS $duration $temperature) 0.0)
   (DEFN PRED1 ($lr $t)
    (is-type silicon (@ (material $lr) $t)))))
(init-form (assert-expr '(FUNCTIONAL EPI-THICKNESS MI MI)))
```

*;;; Abstract parent of all process steps that can modify/destroy existing layers*
*;;; by selectively destroying layer-regions*

```
(defprocess DAS-MODIFY-LAYERS
  effects
  ((for-all-existing $lr \: layer-region (SPI)
    (:IFF (:OR (destroyed $lr (EPI))
               (converted $lr (EPI)))
          (member (@ (layer $lr) (SPI))
                  (changed-layers (PI)))))
   (for-all-true $l e (changed-layers (PI))
    (:AND
     (for-all-true $lr e (@ (layer-regions $l) (SPI))
      (:IFF (:NOT (destroyed $lr (EPI)))
            (member $lr (new-layer-regions $l (PI)))))
     (:IFF
      (for-all $lr e (layer-closure $l (SPI))
       (destroyed $lr (EPI)))
      (destroyed $l (EPI)))
     (:=> (:NOT (destroyed $l (EPI)))
          (change = (layer-regions $l)
            (new-layer-regions $l (PI)))))))
  constraints
  ((DEFN SPI () (start-of (process interval)))
   (DEFN EPI () (end-of (process interval)))
   (DEFN PI  () (process interval))))
```

```
(defprocess DAS-PHOTO-RESIST-CLEAN
  effects
  ((for-all-existing $rt \: region-type (SPI)
    (:=>
     (is-type photoresist
              (@ (material (surface $rt)) (SPI)))
     (:AND
      (destroyed (@ (surface $rt) (SPI)) (EPI))
      (destroyed (@ (layer (surface $rt)) (SPI)) (EPI))
      (change = (surface $rt)
        (old-second-layer-region $rt (PI)))
      (change = (above (old-second-layer-region $rt (PI)))
        THE-ENVIRONMENT)))))
  constraints
  ((DEFN SPI () (start-of (process interval)))
   (DEFN EPI () (end-of (process interval)))
   (DEFN PI  () (process interval))
   (DEFN OLD-SECOND-LAYER-REGION (rt i)
    (@ (below (surface rt)) (start-of i)))))


(defprocess DAS-PHOTO-RESIST-DEVELOP
  AKO DAS-MODIFY-LAYERS
  effects
  ((for-all-existing $ld \: region-type (SPI)
    (:AND
     (:IFF
       (is-soft-photoresist?
         (@ (material (surface $ld)) (SPI)))
       (destroyed (@ (surface $ld) (SPI)) (EPI)))
     (:IF
       (destroyed (@ (surface $ld) (SPI)) (EPI))
       (:AND
        (change = (surface $ld)
          (old-second-layer-region $ld (PI)))
        (change = (above (old-second-layer-region $ld (PI)))
          THE-ENVIRONMENT))))))
  constraints
  ((DEFN SPI () (start-of (process interval)))
   (DEFN EPI () (end-of (process interval)))
   (DEFN PI  () (process interval))
   (DEFN OLD-SECOND-LAYER-REGION (ld i)
     (@ (below (surface ld)) (start-of i)))))
```

```
(defprocess DAS-ETCH
  AKO DAS-MODIFY-LAYERS
  parameters (($etchant etchant-type) ($duration finite-positive-real))
  effects
  ((for-all-existing $l \: layer-region (SPI)
     (:AND
      (:IFF (>= $duration (etch-destroy-time $l (PI)))
            (consumed-by-etching $l (EPI)))
      (:IFF (consumed-by-etching $l (EPI))
            (destroyed $l (EPI)))
      (:IF
       (:AND (>= $duration
                 (etch-destroy-time (@ (above $l) (SPI))
                                   (PI)))
             (:NOT (consumed-by-etching $l (EPI))))
       (:AND
        (:IF
         (:NOT (V= $l (@ (surface (region-type $l)) (SPI))))
         (:AND
          (change = (surface (@ (region-type $l) (SPI))) $l)
          (change = (above $l) THE-ENVIRONMENT)))
        (:IF
         (:AND (> $duration
                  (etch-destroy-time (@ (above $l) (SPI))
                                    (PI)))
               (> (etch-rate $l (PI)) 0.0))
         (:AND (change - (top-pos $l) (amount-etched $l))
               (change - (thickness $l)
                 (amount-etched $l)))))))))
  constraints
  ((DEFN SPI () (start-of (process interval)))
   (DEFN EPI () (end-of (process interval)))
   (DEFN PI  () (process interval))
   (DEFN AMOUNT-ETCHED ($l)
     (* (- $duration
           (etch-destroy-time
              (@ (above $l)(start-of (process interval)))
              (process interval)))
        (etch-rate $l (process interval))))))
```

```
(defprocess DAS-END-DETECT-ETCH
  AKO DAS-Etch
  parameters
  (($overetch finite-non-negative-real)
   ($detected-layer layer)
   ($etch-angle finite-positive-real))
  effects
  ((for-all-true $lr1 e (@ (layer-regions $detected-layer)
                          (start-of (process interval)))
     (:AND
      ;; Detected layer is not etched through
      (< $duration
         (etch-destroy-time $lr1 (process interval)))
      ;; Layer regions "closest" to surface (in terms of etch time) are detected.
      (:IFF
        (for-all $lr2 e (layer-closure $detected-layer
                          (start-of (process interval)))
          (<= (etch-destroy-time
               (@ (above $lr1) (start-of (process interval)))
               (process interval))
              (etch-destroy-time
               (@ (above $lr2) (start-of (process interval)))
               (process interval))))
        (member $lr1
          (detected-layer-regions (process interval))))))
     (N= (vertical-etch-duration (process interval))
         (etch-destroy-time
           (@ (above (any-member (detected-layer-regions
                                  (process interval))))
              (start-of (process interval)))
           (process interval)))
     (N= $duration
         (+ (vertical-etch-duration (process interval))
            (* $overetch
               (vertical-etch-duration
                 (process interval)))))))
```

```
;;; Abstract process for process steps that can create as well as destroy layer-regions.
;;; Newly created layers are associated with existing layers that help define and
;;; differentiate them. The set (new-layers (process interval)) contains the existing
;;; layers that each define a new layer. The sets (new-layer old-layer (process interval))
;;; contain the newly created layer-regions that are to belong to the new layers.
(defprocess DAS-CREATE/MODIFY-LAYERS
  AKO DAS-Modify-Layers
  effects
  ((for-all-true $old-layer e (new-layers (process interval))
     (CREATED ($new-layer layer (start-of (process interval)))
      (for-all-true $lr e (new-layer $old-layer
                          (process interval))
        (:AND
         (member $lr (@ (layer-regions $new-layer)
                        (end-of (process interval))))
         (change = (layer $lr) $new-layer)))))))
```

```
(defprocess DAS-MASK-EXPOSE
  parameters (($mask mask) ($structure 2d-structure))
  effects
  ((change = (horizontal-regions $structure)
           (new-horizontal-regions $structure (PI)))
   (for-all-true $mask-region e (@ (horizontal-regions $mask) (spi))
     (for-all-true $reg e (@ (horizontal-regions $structure) (spi))
      ;; Overlap
      (:IF
       (:AND (< (@ (left-pos $mask-region) (spi))
                (@ (right-pos $reg) (spi)))
             (> (@ (right-pos $mask-region) (spi))
                (@ (left-pos $reg) (spi))))
       (:AND
        ;; Complete Overlap
        (:IF
         (:AND
           (<= (@ (left-pos $mask-region) (spi))
               (@ (left-pos $reg) (spi)))
           (>= (@ (right-pos $mask-region) (spi))
               (@ (right-pos $reg) (spi))))
         (:AND
          (member $reg (new-horizontal-regions $structure (PI)))
          (:IF
           (is-type photoresist
            (@ (material (surface (region-type $reg))) (spi)))
           (:AND
            (:IF
             (v= (@ (opacity $mask-region) (spi))
                 (constant transparent))
             (:AND
              (member (@ (region-type $reg) (spi))
                (ALTERED-REGION-TYPES (PI)))
              (member $reg
                (ALTERED-REGIONS (@ (region-type $reg) (spi)) (PI)))))
            (:IF
             (v= (@ (opacity $mask-region) (spi))
                 (constant opaque))
             (:AND (member (@ (region-type $reg) (spi))
                      (UNALTERED-REGION-TYPES (process interval)))
                   (member $reg
                    (UNALTERED-REGIONS
                      (@ (region-type $reg) (spi)) (PI)))))))
          (:IF
           (n= (@ (left-pos $mask-region) (spi))
               (@ (left-pos $reg) (spi)))
           (:AND
            (member $reg
              (CHANGING-NEIGHBOUR-WAFER-REGIONS (spi)))
            (change = (left-wafer-region $mask-region) $reg)))
          (:IF
           (n= (@ (right-pos $mask-region) (spi))
               (@ (right-pos $reg) (spi)))
           (:AND
            (member $reg (CHANGING-NEIGHBOUR-WAFER-REGIONS (spi)))
            (change = (right-wafer-region $mask-region) $reg)))
          (:IF
           (:NOT (v= (@ (left $reg) (spi)) REGION-EDGE))
```

```
   (:IF
    (:AND (< (@ (left-pos $mask-region) (spi))
             (@ (left-pos $reg) (spi)))
          (>= (@ (left-pos $mask-region) (spi))
              (@ (left-pos (left $reg)) (spi))))
    (member $reg (CHANGING-NEIGHBOUR-WAFER-REGIONS (spi)))))
  (:IF
   (:NOT (v= (@ (right $reg) (spi)) REGION-EDGE))
   (:IF
    (:AND (> (@ (right-pos $mask-region) (spi))
             (@ (right-pos $reg) (spi)))
          (<= (@ (right-pos $mask-region) (spi))
              (@ (right-pos (right $reg)) (spi))))
    (member $reg (CHANGING-NEIGHBOUR-WAFER-REGIONS (spi)))))))
;; Partial Overlap
(:IF
 (:OR (> (@ (left-pos $mask-region) (spi))
         (@ (left-pos $reg) (spi)))
      (< (@ (right-pos $mask-region) (spi))
         (@ (right-pos $reg) (spi))))
 (CREATED ($new-reg wafer-region (spi))
  (change = (pieces $reg) (NEW-PIECES $reg (spi)))
  (member $new-reg (NEW-PIECES $reg (spi)))
  (member $new-reg (new-horizontal-regions $structure (PI)))
  (member $new-reg (CHANGING-NEIGHBOUR-WAFER-REGIONS (spi)))
  (change = (pieces $new-reg) NULL-SET)
  (:IF
   (:AND
    (is-type photoresist
      (@ (material (surface (region-type $reg))) (spi)))
    (v= (@ (opacity $mask-region) (spi))
        (constant transparent)))
   (:AND
    (member (@ (region-type $reg) (spi))
      (ALTERED-REGION-TYPES (process interval)))
    (member $new-reg
      (ALTERED-REGIONS (@ (region-type $reg) (spi)) (PI)))))
  (:IF
   (:NOT
    (:AND (is-type photoresist
            (@ (material (surface (region-type $reg))) (spi)))
          (v= (@ (opacity $mask-region) (spi))
              (constant transparent))))
   (:AND (member (@ (region-type $reg) (spi))
          (UNALTERED-REGION-TYPES (process interval)))
         (member $new-reg
          (UNALTERED-REGIONS (@ (region-type $reg) (spi)) (PI)))
         (change = (region-type $new-reg)
           (@ (region-type $reg) (spi)))))
  ;; Set Up Right Pointers And Right-Pos
  (:IFF
   (:NOT (V= (@ (right $reg) (spi)) REGION-EDGE))
   (:IF
    (> (@ (right-pos $mask-region) (spi))
       (@ (right-pos $reg) (spi)))
    (:AND
     (change = (right-pos $new-reg) (@ (right-pos $reg) (spi)))
     (:IF (> (@ (right-pos $mask-region) (spi))
```

```
                (@ (right-pos (right $reg)) (spi)))
             (change = (right $new-reg) (@ (right $reg) (spi)))))))
     (:IF
      (<= (@ (right-pos $mask-region) (spi))
          (@ (right-pos $reg) (spi)))
      (:AND
       (change = (right-pos $new-reg)
         (@ (right-pos $mask-region) (spi)))
       (change = (right-wafer-region $mask-region) $new-reg)))
```

*;; Set Up Left Pointers And Left-Pos*

```
     (:IFF
      (:NOT (V= (@ (left $reg) (spi)) REGION-EDGE))
      (:IF
       (< (@ (left-pos $mask-region) (spi))
          (@ (left-pos $reg) (spi)))
       (:AND
        (change = (left-pos $new-reg) (@ (left-pos $reg) (spi)))
        (:IF (< (@ (left-pos $mask-region) (spi))
                (@ (left-pos (left $reg)) (spi)))
             (change = (left $new-reg) (@ (left $reg) (spi)))))))
     (:IF
      (>= (@ (left-pos $mask-region) (spi))
          (@ (left-pos $reg) (spi)))
      (:AND
       (change = (left-pos $new-reg)
         (@ (left-pos $mask-region) (spi)))
       (change = (left-wafer-region $mask-region) $new-reg)))
      ))))))


(for-all-true $reg e (CHANGING-NEIGHBOUR-WAFER-REGIONS (spi))
  (for-all-true $mask-region e (@ (horizontal-regions $mask) (spi))
    (:AND
     (:IF
      (v= $reg (@ (left-wafer-region $mask-region) (epi)))
      (:AND
       (:IFF (:NOT (v= (@ (left $mask-region) (spi)) MASK-EDGE))
             (change = (left $reg)
               (@ (right-wafer-region (left $mask-region)) (epi))))
       (:IF (v= (@ (left $mask-region) (epi)) MASK-EDGE)
            (change = (left $reg) REGION-EDGE))))
     (:IF
      (v= $reg (@ (right-wafer-region $mask-region) (epi)))
      (:AND
       (:IFF (:NOT (v= (@ (right $mask-region) (spi)) MASK-EDGE))
             (change = (right $reg)
               (@ (left-wafer-region (right $mask-region)) (epi))))
       (:IF (v= (@ (right $mask-region) (epi)) MASK-EDGE)
            (change = (right $reg) REGION-EDGE))))
     (:IF
      (n= (@ (right-pos (left-wafer-region $mask-region)) (epi))
          (@ (left-pos $reg) (epi)))
      (change = (left $reg)
        (@ (left-wafer-region $mask-region) (epi))))
     (:IF
      (n= (@ (left-pos (right-wafer-region $mask-region)) (epi))
          (@ (right-pos $reg) (epi)))
      (change = (right $reg)
        (@ (right-wafer-region $mask-region) (epi)))))))
```

```
 (for-all-true $rt e (ALTERED-REGION-TYPES (PI))
  (:IF
   (is-type photoresist (@ (material (surface $rt)) (spi)))
   (:AND
    (:IF (member $rt (UNALTERED-REGION-TYPES (PI)))
         (CREATED ($new-rt region-type (spi))
           (change = (surface $new-rt)
             (EXPOSED-LAYER-REGION (@ (surface $rt) (epi)) (PI)))
           (v= (EXPOSED-REGION-TYPE $rt (process interval)) $new-rt)
           (change = (regions $new-rt) (ALTERED-REGIONS $rt (PI)))
           (for-all-true $reg e (ALTERED-REGIONS $rt (PI))
             (change = (region-type $reg) $new-rt))
           (change = (regions $rt) (UNALTERED-REGIONS $rt (PI)))))
    (:IF (:NOT (member $rt (UNALTERED-REGION-TYPES-CLOSURE (PI))))
         (change = (material (surface $rt))
           (EXPOSED-MATERIAL (@ (material (surface $rt)) (spi))))))))
 (for-all-existing $l \: layer-region (spi)
   (:IF
    (:AND
     (is-type photoresist
        (@ (material (surface (region-type $l))) (spi)))
     (member (@ (region-type $l) (spi))
       (ALTERED-REGION-TYPES (PI)))
     (member (@ (region-type $l) (spi))
       (UNALTERED-REGION-TYPES (PI))))
    (CREATED ($new-l layer-region (spi))
     (change = (layer-regions (layer $l))
       (new-layer-regions (@ (layer $l) (epi)) (PI)))
     (member $new-l
       (new-layer-regions (@ (layer $l) (epi)) (PI)))
     (change = (layer $new-l) (@ (layer $l) (epi)))
     (change = (region-type $new-l)
       (EXPOSED-REGION-TYPE (@ (region-type $l) (spi)) (pi)))
     (change = (top-pos $new-l) (@ (top-pos $l) (spi)))
     (:IF (v= (@ (above $l) (spi)) THE-ENVIRONMENT)
          (change = (above $new-l) (@ (above $l) (spi))))
     (:IF (:NOT (v= THE-ENVIRONMENT (@ (above $l) (spi))))
          (change = (above $new-l)
            (EXPOSED-LAYER-REGION (@ (above $l) (spi)) (pi))))
     (change = (bottom-pos $new-l) (@ (bottom-pos $l) (spi)))
     (:IF (:NOT (v= (@ (below $l) (spi)) THE-ENVIRONMENT))
          (change = (below $new-l)
            (EXPOSED-LAYER-REGION (@ (below $l) (spi)) (pi))))
     (:IF (v= (@ (below $l) (spi)) THE-ENVIRONMENT)
          (change = (below $new-l) THE-ENVIRONMENT))
     (change = (material $new-l)
       (EXPOSED-MATERIAL (@ (material $l) (spi))))
     (change = (dopant $new-l) (@ (dopant $l) (spi)))
     (change = (concentration $new-l) (@ (concentration $l) (spi)))
     (change = (thickness $new-l) (@ (thickness $l) (spi)))
     (v= (EXPOSED-LAYER-REGION $l (process interval)) $new-l))))
 (for-all-existing $l |:| layer (spi)
   (for-all-true $lr e (@ (layer-regions $l) (spi))
     (member $lr (@ (layer-regions $l) (epi))))))
constraints ((DEFN PI ()  (process interval))
             (DEFN SPI () (start-of (process interval)))
             (DEFN EPI () (end-of (process interval)))))
```

```
(defprocess DAS-PRE-DEPOSITION
  AKO DAS-Create/Modify-Layers
  parameters
  (($dopant dopant) ($duration finite-positive-real)
  ($temperature finite-positive-real)
  ($vapor-pressure finite-positive-real))
  effects
  ((for-all-existing $rt \: region-type (SPI)
     (:=>
      (pred1 (@ (surface $rt) (SPI))
             (SPI))
      (CREATED ($NL layer-region (SPI))
       (member $NL (new-layer (@ (layer (surface $RT)) (SPI)) (PI)))
       (member (@ (layer (surface $RT)) (SPI)) (new-layers (PI)))
       (change = (surface $rt) $nl)
       (change = (region-type $nl) $rt)
       (change = (top-pos $nl) (old-top-pos $rt (PI)))
       (change = (above $nl) THE-ENVIRONMENT)
       (change - (top-pos (old-top $rt (PI))) (mat-thick $rt (PI)))
       (change - (thickness (old-top $rt (PI))) (mat-thick $rt (PI)))
       (change = (bottom-pos $nl) (- (old-top-pos $rt (PI))
                                     (mat-thick $rt (PI))))
       (change = (below $nl) (old-top $rt (PI)))
       (change = (dopant $nl) $dopant)
       (change = (concentration $nl)
               (PRE-DEP-CONC $vapor-pressure
                             (old-material $rt (PI))
                             $dopant))
       (change = (material $nl) (old-material $rt (PI)))
       (change = (thickness $nl) (mat-thick $rt (PI)))
       (change = (above (old-top $rt (PI))) $nl)))))
  constraints
  ((DEFN PI () (process interval))
   (DEFN SPI () (start-of (process interval)))
   (DEFN OLD-TOP (rt i) (@ (surface rt) (start-of i)))
   (DEFN OLD-TOP-POS (rt i) (@ (top-pos (surface rt)) (start-of i)))
   (DEFN OLD-MATERIAL (rt i) (@ (material (surface rt)) (start-of i)))
   (DEFN MAT-THICK (rt i)
     (PRE-DEP-THICKNESS (@ (material (surface rt)) (start-of i))
                        $dopant $duration $temperature))
   (for-all-existing $rt \: region-type (start-of (process interval))
    (:=>
     (Accepts-Dopant?
      (@ (material (surface $rt)) (start-of (process interval)))
      $dopant)
    (> (PRE-DEP-THICKNESS
        (@ (material (surface $rt)) (start-of (process interval)))
        $dopant $duration $temperature)
       0.0)))
   (DEFN PRED1 ($lr $t)
     (Accepts-Dopant? (@ (material $lr) $t) $dopant))))
(init-form
  (assert-expr '(FUNCTIONAL PRE-DEP-THICKNESS NA NA MI MI)))
```

```
(defprocess DAS-ION-IMPLANTATION
  AKO DAS-Create/Modify-Layers
  parameters
  (($dopant dopant) ($duration finite-positive-real)
   ($implant-energy finite-positive-real)
   ($dose finite-positive-real))
  effects
  ((for-all-existing $l \: layer-region (SPI)
    ;; Any overlap of implant span with this layer-region?
    (:IF
     (:AND (> (it) (lb)) (< (ib) (lt)))
     (:AND

      ;; Complete Overlap (make gross simplification that photo resist and
      ;;    polysilicon always are totally implanted, if implanted at all)
      (:IF (:OR (is-type photoresist (@ (material $l) (spi)))
                (is-type polysilicon (@ (material $l) (spi)))
                (:AND (<= (ib) (lb)) (>= (it) (lt))))
           (:AND (dopant-concentration-change $l $l)
                 (member $l  (new-layer (@ (layer $l) (spi)) (PI)))
                 (member (@ (layer $l) (spi)) (new-layers (PI)))
                 (converted $l (epi))))

      ;; Partial Overlap From Top
      (:IF
       (:AND (> (ib) (lb)) (>= (it) (lt)))
       (CREATED ($nl layer-region (spi))
        (member $nl (new-layer (@ (layer $l) (spi)) (PI)))
        (member (@ (layer $l) (spi)) (new-layers (PI)))
        (dopant-concentration-change $nl $l)
        (change = (thickness $nl) (- (lt) (ib)))
        (change - (thickness $l) (- (lt) (ib)))
        (change = (top-pos $nl) (@ (top-pos $l) (spi)))
        (change = (top-pos $l) (ib))
        (change = (bottom-pos $nl) (ib))
        (change = (material $nl) (@ (material $l) (spi)))
        (change = (region-type $nl) (@ (region-type $l) (spi)))

        ;; Partially stitch together layer-regions
        (:IF (v= (@ (above $l) (spi)) THE-ENVIRONMENT)
             (:AND (change = (above $nl) THE-ENVIRONMENT)
                   (change = (surface (@ (region-type $l) (spi))) $nl)))
        (:IF (:AND (:NOT (v= (@ (above $l) (spi)) THE-ENVIRONMENT))
                   (n= (it) (lt)))
             (:AND (change = (above $nl) (@ (above $l) (spi)))
                   (change = (below (above $l)) $nl)))
        (:IF (> (it) (lt))
             (member $l (overlapped-from-top-layer-regions (PI))))
        (change = (above $l) $nl)
        (change = (below $nl) $l)))

      ;; Partial Overlap From Below
      (:IF
       (:AND (< (it) (lt)) (<= (ib) (lb)))
       (CREATED ($nl layer-region (spi))
```

```
      (member $nl (new-layer (@ (layer $l) (spi)) (PI)))
      (member (@ (layer $l) (spi)) (new-layers (PI)))
      (dopant-concentration-change $nl $l)
      (change = (thickness $nl) (- (it) (lb)))
      (change - (thickness $l) (- (it) (lb)))
      (change = (bottom-pos $nl) (@ (bottom-pos $l) (spi)))
      (change = (bottom-pos $l) (it))
      (change = (top-pos $nl) (it))

      ;; Partially stitch together layer-regions
      (:IF (v= (@ (below $l) (spi)) THE-ENVIRONMENT)
           (change = (below $nl) THE-ENVIRONMENT))
      (:IF (:AND
            (:NOT (v= (@ (below $l) (spi)) THE-ENVIRONMENT))
            (n= (ib) (lb)))
           (:AND (change = (below $nl) (@ (below $l) (spi)))
                 (change = (above (@ (below $l) (spi))) $nl)))
      (:IF (< (ib) (lb))
           (member $l  (overlapped-from-bottom-layer-regions (PI))))
      (change = (below $l) $nl)
      (change = (above $nl) $l)
      (change = (material $nl) (@ (material $l) (spi)))
      (change = (region-type $nl) (@ (region-type $l) (spi)))))

    ;; Complete Containment Within A Layer (Most Likely Case?)
    (:IF
     (:AND (> (ib) (lb)) (< (it) (lt)))
     ;; "IMP" is the new implanted layer-region,
     ;; "SPLIT" is split area below "IMP"
     (CREATED ($split layer-region (spi))
       (change = (layer $split) (@ (layer $l) (spi)))
       (change = (dopant $split) (@ (dopant $l) (spi)))
       (change = (concentration $split)
         (@ (concentration $l) (spi)))
       (change = (thickness $split) (- (ib) (lb)))
       (change = (top-pos $split) (ib))
       (change = (bottom-pos $split) (lb))
       (change = (below $split) (@ (below $l) (spi)))
       (change = (material $split) (@ (material $l) (spi)))
       (change = (region-type $split)
         (@ (region-type $l) (spi)))
       (:IF (:NOT (v= (@ (below $l) (spi)) THE-ENVIRONMENT))
            (change = (above (@ (below $l) (spi))) $split))
       (CREATED ($imp layer-region (spi))
        (member $imp (new-layer (@ (layer $l) (spi)) (PI)))
        (member (@ (layer $l) (spi)) (new-layers (PI)))
        (dopant-concentration-change $imp $l)
        (change = (thickness $imp) (- (it) (ib)))
        (change = (thickness $l) (- (lt) (it)))
        (change = (bottom-pos $imp) (ib))
        (change = (below $imp) $split)
        (change = (above $split) $imp)
        (change = (bottom-pos $l) (it))
        (change = (below $l) $imp)
        (change = (top-pos $imp) (it))
        (change = (above $imp) $l)
        (change = (material $imp) (@ (material $l) (spi)))
```

```
          (change = (region-type $imp) (@ (region-type $l) (spi)))))))))

   ;; Finish stitching together layer-regions
   (for-all-true $lr e  (overlapped-from-top-layer-regions (PI))
    (:AND
     (:IF (member (@ (above $lr) (spi))
            (overlapped-from-bottom-layer-regions (PI)))
          (change = (above (@ (above $lr) (EPI)))
            (@ (below (@ (above $lr) (SPI))) (EPI))))
     (:IF (converted (@ (above $lr) (SPI)) (epi))
          (:AND
           (change = (above (@ (above $lr) (EPI)))
                   (@ (above $lr) (SPI)))
           (change = (below (above $lr))
                   (@ (above $lr) (EPI)))))))
   (for-all-true $lr e (overlapped-from-bottom-layer-regions (PI))
    (:AND
     (:IF (member (@ (below $lr) (spi))
                  (overlapped-from-top-layer-regions (PI)))
          (change = (below (@ (below $lr) (EPI)))
                  (@ (above (@ (below $lr) (SPI))) (EPI))))
     (:IF (converted (@ (below $lr) (SPI)) (epi))
          (:AND
           (change = (below (@ (below $lr)(EPI))) (@ (below $lr) (SPI)))
           (change = (above (below $lr)) (@ (below $lr) (EPI))))))))
  constraints
  ((DEFN PI () (process interval))
   (DEFN SPI () (start-of (process interval)))
   (DEFN EPI () (end-of (process interval)))
   (DEFN IT ()
     (ion-top (@ (region-type $l) (start-of (process interval)))
              (process interval)))
   (DEFN IB ()
     (ion-bottom (@ (region-type $l) (start-of (process interval)))
                 (process interval)))
   (DEFN LT () (@ (top-pos $l) (start-of (process interval))))
   (DEFN LB () (@ (bottom-pos $l) (start-of (process interval))))
   (for-all-existing $l \: layer-region (start-of (process interval))
    (:=>
     (:AND
      (>= $implant-energy
          (transmission-energy
            (@ (above $l) (start-of (process interval)))
            (start-of (process interval))))
      (< $implant-energy
         (transmission-energy $l (start-of (process interval)))))
     (:AND
      (n= (median-implant-distance
            (@ (region-type $l) (spi)) (pi))
          (+ (- (@ (top-pos (surface (region-type $l)))
                   (start-of (process interval)))
                (@ (top-pos $l) (spi)))
             (* (- $implant-energy
                   (transmission-energy
                     (@ (above $l) (start-of (process interval)))
                     (start-of (process interval))))
                (energy-loss-rate
                 (@ (material $l) (start-of (process interval)))))))
```

```
    (<= (median-implant-position (@ (region-type $l) (spi)) (pi))
        (@ (top-pos $l) (spi)))
    (>  (median-implant-position (@ (region-type $l) (spi)) (pi))
        (@ (bottom-pos $l) (spi))))))
(DEFN DOPANT-CONCENTRATION-CHANGE ($nl $l)
  (:AND
    (:=>
     (v= (@ (dopant $l) (start-of (process interval))) *NO-DOPANT*)
     (:AND (change = (dopant $nl) $dopant)
           (change = (concentration $nl)
             (implanted-conc
               (@ (region-type $l) (start-of (process interval)))
               (process interval))))))
    (:=>
     (v= (@ (dopant $l) (start-of (process interval)))
         $dopant)
     (:AND
      (:=> (:NOT (v= $l $nl))
           (change = (dopant $nl)
             (@ (dopant $l) (start-of (process interval)))))
      (change = (concentration $nl)
        (+ (@ (concentration $l) (start-of (process interval)))
           (implanted-conc
             (@ (region-type $l) (start-of (process interval)))
           (process interval))))))
    (:=>
     (:NOT (:OR (v= (@ (dopant $l) (start-of (process interval)))
                    *NO-DOPANT*)
                (v= (@ (dopant $l) (start-of (process interval)))
                    $dopant)))
     (:AND
      (:=>
       (> (@ (concentration $l)
             (start-of (process interval)))
          (implanted-conc
            (@ (region-type $l)
               (start-of (process interval)))
            (process interval)))
       (:AND
        (change = (concentration $nl)
          (- (@ (concentration $l)
                (start-of (process interval)))
             (implanted-conc
               (@ (region-type $l)
                  (start-of (process interval)))
               (process interval))))
        (change = (dopant $nl)
          (@ (dopant $l)
             (start-of (process interval))))))
      (:=> (:NOT (> (@ (concentration $l)
                       (start-of (process interval)))
                    (implanted-conc
                      (@ (region-type $l)
                         (start-of (process interval)))
                      (process interval))))
           (:AND
            (change = (dopant $nl) $dopant)
            (change = (concentration $nl)
```

```
                (- (implanted-conc
                     (@ (region-type $l)
                        (start-of (process interval)))
                     (process interval))
                   (@ (concentration $l)
                      (start-of (process interval)))))))
            ))))))


(defprocess DAS-DIFFUSION
  AKO DAS-Create/Modify-Layers
  parameters
  (($temperature finite-positive-real)
   ($duration finite-positive-real))
  preconds ((> $temperature MINIMUM-DIFFUSION-TEMPERATURE))
  effects
  ((for-all-existing $l \: layer-region (spi)
    (:AND
     (:IFF (consumed-by-diffusion $l (epi))
           (destroyed $l (epi)))
     ;; Diffusion from/to the top only--same basic material
     (:IF
      (:AND
       (v= (@ (material $l) (spi))
           (@ (material (above $l)) (spi)))
       (:OR (v= (@ (below $l) (spi)) THE-ENVIRONMENT)
            (:NOT (v= (@ (material $l) (spi))
                      (@ (material (below $l)) (spi))))))
      (:AND
       (:IFF (<= (@ (thickness $l) (spi))
                 (diffusion-distance
                   $duration $temperature
                   (@ (concentration (above $l)) (spi))
                   (@ (concentration $l) (spi))))
             (consumed-by-diffusion $l (epi)))
       (:IF  (consumed-by-diffusion $l (epi))
             (:AND
              (change = (below (above $l))
                (@ (below $l) (spi)))
              (:IF (:NOT (V= (@ (below $l) (spi))
                             THE-ENVIRONMENT))
                   (change = (above (below $l))
                     (@ (above $l) (spi))))))
       (:IF
        (:AND
         (:NOT (consumed-by-diffusion $l (epi)))
         (<> 0.0 (diffusion-distance
                    $duration $temperature
                    (@ (concentration (above $l)) (spi))
                    (@ (concentration $l) (spi)))))
        (:AND
         (change - (thickness $l)
           (diffusion-distance
              $duration $temperature
              (@ (concentration (above $l)) (spi))
              (@ (concentration $l) (spi))))
         (change - (top-pos $l)
           (diffusion-distance
              $duration $temperature
```

```
          (@ (concentration (above $l)) (spi))
          (@ (concentration $l) (spi))))
     (:=> (> (@ (concentration $l) (spi))
             (@ (concentration (above $l)) (spi)))
          (change - (concentration $l)
            (diffusion-depletion
               $duration $temperature
               (@ (concentration $l) (spi))
               (@ (concentration (above $l)) (spi)))
          ))))))
```

*;; Diffusion from/to the bottom only--same basic material*

```
(:IF
 (:AND
  (v= (@ (material $l) (spi))
      (@ (material (below $l)) (spi)))
  (:OR (v= (@ (above $l) (spi)) THE-ENVIRONMENT)
       (:NOT (v= (@ (material $l) (spi))
                 (@ (material (above $l)) (spi))))))
 (:AND
  (:IFF
   (:NOT (> (@ (thickness $l) (spi))
            (diffusion-distance $duration $temperature
               (@ (concentration (below $l)) (spi))
               (@ (concentration $l) (spi)))))
   (consumed-by-diffusion $l (epi)))
  (:IF
   (consumed-by-diffusion $l (epi))
   (:AND
    (:IF (V= $l (@ (surface (region-type $l)) (spi)))
         (change = (surface (@ (region-type $l) (spi)))
                 (@ (below $l) (spi))))
    (change = (above (below $l)) (@ (above $l) (spi)))
    (:IF (:NOT (V= (@ (above $l) (spi))
                   THE-ENVIRONMENT))
         (change = (below (above $l))
           (@ (below $l) (spi))))))
  (:IF
   (:AND (:NOT (consumed-by-diffusion $l (epi)))
         (<> 0.0 (diffusion-distance $duration
                   $temperature
                   (@ (concentration (below $l)) (spi))
                   (@ (concentration $l) (spi)))))
   (:AND
    (change - (thickness $l)
      (diffusion-distance $duration $temperature
        (@ (concentration (below $l)) (spi))
        (@ (concentration $l) (spi))))
    (change + (bottom-pos $l)
      (diffusion-distance $duration $temperature
        (@ (concentration (below $l)) (spi))
        (@ (concentration $l) (spi))))
    (:IF (> (@ (concentration $l) (spi))
            (@ (concentration (below $l)) (spi)))
         (change - (concentration $l)
           (diffusion-depletion $duration $temperature
             (@ (concentration $l) (spi))
             (@ (concentration (below $l)) (spi))))
         )))))
```

*;; Diffusion from/to both top and bottom--same basic material*

```
(:IF
 (:AND (v= (@ (material $l) (spi))
           (@ (material (below $l)) (spi)))
       (v= (@ (material $l) (spi))
           (@ (material (above $l)) (spi))))
 (:AND
  (:IFF
   (:NOT
    (> (@ (thickness $l) (spi))
       (+ (diffusion-distance $duration $temperature
            (@ (concentration (above $l)) (spi))
            (@ (concentration $l) (spi)))
          (diffusion-distance $duration $temperature
            (@ (concentration (below $l)) (spi))
            (@ (concentration $l) (spi))))))
    (consumed-by-diffusion $l (epi)))
  (:IF  (consumed-by-diffusion $l (epi))
        (:AND
         (change = (above (below $l))
           (@ (above $l) (spi)))
         (change = (below (above $l))
           (@ (below $l) (spi)))))
  (:IF
   (:AND
    (> (@ (thickness $l) (spi))
       (+ (diffusion-distance $duration $temperature
             (@ (concentration (above $l)) (spi))
             (@ (concentration $l) (spi)))
          (diffusion-distance $duration $temperature
             (@ (concentration (below $l)) (spi))
             (@ (concentration $l) (spi)))))
    (:OR (<> 0.0 (diffusion-distance $duration
                   $temperature
                   (@ (concentration (above $l)) (spi))
                   (@ (concentration $l) (spi))))
         (<> 0.0 (diffusion-distance $duration
                   $temperature
                   (@ (concentration (below $l)) (spi))
                   (@ (concentration $l) (spi))))))
   (:AND
    (change - (thickness $l)
            (+ (diffusion-distance $duration
                 $temperature
                 (@ (concentration (above $l)) (spi))
                 (@ (concentration $l) (spi)))
               (diffusion-distance $duration
                 $temperature
                 (@ (concentration (below $l)) (spi))
                 (@ (concentration $l) (spi)))))
    (:=> (<> 0.0 (diffusion-distance $duration
                   $temperature
                   (@ (concentration (above $l)) (spi))
                   (@ (concentration $l) (spi))))
         (change - (top-pos $l)
                 (diffusion-distance $duration
                   $temperature
                   (@ (concentration (above $l)) (spi))
```

```
                      (@ (concentration $l) (spi)))))
         (:=> (<> 0.0 (diffusion-distance $duration
                      $temperature
                      (@ (concentration (below $l)) (spi))
                      (@ (concentration $l) (spi))))
              (change + (bottom-pos $l)
                      (diffusion-distance $duration
                      $temperature
                      (@ (concentration (below $l)) (spi))
                      (@ (concentration $l) (spi)))))
         (:=> (:OR (> (@ (concentration $l) (spi))
                      (@ (concentration (above $l)) (spi)))
                   (> (@ (concentration $l) (spi))
                      (@ (concentration (below $l)) (spi))))
              (change - (concentration $l)
                (+ (diffusion-depletion
                     $duration $temperature
                     (@ (concentration $l) (spi))
                     (@ (concentration (above $l)) (spi)))
                   (diffusion-depletion
                     $duration $temperature
                     (@ (concentration $l) (spi))
                     (@ (concentration (below $l)) (spi))))
                )))))))))
constraints
((for-all-existing $l \: layer-region (epi)
   (:AND
    (:=> (>  (@ (concentration $l) (spi))
             (@ (concentration (above $l)) (spi)))
         (>= (@ (concentration $l) (epi))
             (@ (concentration (above $l)) (epi))))
    (:=> (<  (@ (concentration $l) (spi))
             (@ (concentration (above $l)) (spi)))
         (<= (@ (concentration $l) (epi))
             (@ (concentration (above $l)) (epi))))
    (n= (@ (top-pos $l) (epi))
        (@ (bottom-pos (above $l)) (epi)))))
 (DEFN SPI () (start-of (process interval)))
 (DEFN EPI () (end-of (process interval)))))
```

```
;; Simultaneous oxidation and diffusion, barring diffusion-induced topological
;; changes.
(defprocess DAS-OXIDATION
  AKO DAS-Create/Modify-Layers
  parameters (($temperature finite-positive-real)
              ($duration finite-positive-real)
              ($ox-duration finite-positive-real))
  preconds ((> $temperature MINIMUM-DIFFUSION-TEMPERATURE))
  effects
  (;; Layer-regions can be consumed by oxidation only
   (for-all-existing $l \: layer-region (spi)
    (:IFF (destroyed $l (epi))
          (consumed-by-oxidation $l (epi))))


   ;; If the surface layer oxidizes, there will be an oxide at the
   ;; surface at the end of the process
   (for-all-existing $rt \: region-type (spi)
    (:=>
     (:AND
      (:NOT (is-type oxide
              (@ (material (surface $rt)) (spi))))
      (> (oxidation-rate $temperature
           (@ (material (surface $rt)) (spi)) 0.0)
         0.0))
     (CREATED
      ($oxide-layer-region layer-region (spi))
      (member $oxide-layer-region
        (new-layer (@ (layer (surface $rt)) (spi)) (PI)))
      (member (@ (layer (surface $rt)) (spi))
        (new-layers (process interval)))
      (change = (surface $rt) $oxide-layer-region)
      (change = (region-type $oxide-layer-region) $rt)
      (change = (above $oxide-layer-region) THE-ENVIRONMENT)
      (change = (material $oxide-layer-region) |Oxide|)
      (change = (dopant $oxide-layer-region) *NO-DOPANT*))))


   (for-all-existing $l \: layer-region (spi)
    (:=>
     (:NOT (is-type oxide (@ (material $l) (spi))))
     (:AND
      ;; Oxidation effects
      (:IFF
       (>= $ox-duration
           (oxidation-destroy-time $l $temperature (spi)))
       (consumed-by-oxidation $l (epi)))


      (:IFF
       (:AND
        (>= $ox-duration
            (oxidation-destroy-time (@ (above $l) (spi))
              $temperature (spi)))
        (:NOT (consumed-by-oxidation $l (epi))))
       (uppermost-oxidation-survivor $l (process interval)))

      ;; $l is uppermost layer that survives oxidation
```

```
(:IF
 (uppermost-oxidation-survivor $l (process interval))
 ;; Oxidation without diffusion (no diffusion between materials)
 (:IF
  (:OR (v= (@ (dopant $l) (spi)) *no-dopant*)
       (:NOT (v= (@ (material $l) (spi))
                 (@ (material (below $l)) (spi)))))
  (:IF
   (:AND
    (> $ox-duration
       (oxidation-destroy-time (@ (above $l) (spi))
         $temperature (spi)))
    (> (oxidation-rate $temperature
         (@ (material $l) (spi))
         (oxide-thickness-above
           (@ (above $l) (spi)) (spi)))
       0.0))
   (change - (top-pos $l) (amount-oxidized $l)))))


(:IF
 (:NOT (consumed-by-oxidation $l (epi)))
 (:AND
  ;; Diffusion effects
  ;; Diffusion from/to the top only--same basic material (no oxidation)
  (:IF
   (:AND
    (:NOT (uppermost-oxidation-survivor $l (PI)))
    (v= (@ (material $l) (spi))
        (@ (material (above $l)) (spi)))
    (:OR (v= (@ (below $l) (spi)) THE-ENVIRONMENT)
         (:NOT (v= (@ (material $l) (spi))
                   (@ (material (below $l)) (spi))))))
   (:AND
    (:=>
     (<> 0.0 (diffusion-distance $duration $temperature
               (@ (concentration (above $l)) (spi))
               (@ (concentration $l) (spi))))
     (:AND
      (change - (top-pos $l)
        (diffusion-distance $duration $temperature
          (@ (concentration (above $l)) (spi))
          (@ (concentration $l) (spi))))
      (:IF
       (> (@ (concentration $l) (spi))
          (@ (concentration (above $l)) (spi)))
       (change - (concentration $l)
         (diffusion-depletion $duration $temperature
           (@ (concentration $l) (spi))
           (@ (concentration (above $l)) (spi)))
          ))))))

  ;; Diffusion from/to the bottom only--
  ;; same basic material (oxidation may be involved)
  (:IF
   (:AND
    (:NOT (v= (@ (below $l) (spi)) THE-ENVIRONMENT))
    (v= (@ (material $l) (spi))
```

```
      (@ (material (below $l)) (spi)))
  (:OR (v= (@ (above $l) (spi)) THE-ENVIRONMENT)
       (:NOT (v= (@ (material $l) (spi))
                 (@ (material (above $l)) (spi))))
       (uppermost-oxidation-survivor $l (PI))))
 (:AND
  (:IFF (:NOT (uppermost-oxidation-survivor $l (PI)))
        (N= (oxidation-reduced-thickness $l (spi))
            (@ (thickness $l) (spi))))
  (:IFF (uppermost-oxidation-survivor $l (PI))
        (N= (oxidation-reduced-thickness $l (spi))
            (- (@ (thickness $l) (spi))
               (amount-oxidized $l))))
  (:IF (uppermost-oxidation-survivor $l (PI))
       (change - (top-pos $l) (amount-oxidized $l)))


  (:IF
   (<> 0.0
       (diffusion-distance $duration $temperature
          (@ (concentration (below $l)) (spi))
          (@ (concentration $l) (spi))))
   (:AND
    (change + (bottom-pos $l)
      (diffusion-distance $duration $temperature
        (@ (concentration (below $l)) (spi))
        (@ (concentration $l) (spi))))
    (:IF (> (@ (concentration $l) (spi))
            (@ (concentration (below $l)) (spi)))
         (change - (concentration $l)
           (diffusion-depletion $duration
             $temperature
             (@ (concentration $l) (spi))
             (@ (concentration (below $l)) (spi))))
     )))))
```

***;; Diffusion from/to both top and bottom--***
***;; same basic material (no oxidation)***

```
(:IF
 (:AND
  (:NOT (v= (@ (above $l) (spi)) THE-ENVIRONMENT))
  (:NOT (v= (@ (below $l) (spi)) THE-ENVIRONMENT))
  (v= (@ (material $l) (spi))
      (@ (material (below $l)) (spi)))
  (v= (@ (material $l) (spi))
      (@ (material (above $l)) (spi)))
  (:NOT (uppermost-oxidation-survivor $l (PI))))
 (:IF
  (:OR
   (<> 0.0
       (diffusion-distance $duration $temperature
         (@ (concentration (above $l)) (spi))
         (@ (concentration $l) (spi))))
   (<> 0.0
       (diffusion-distance $duration $temperature
         (@ (concentration (below $l)) (spi))
         (@ (concentration $l) (spi)))))
  (:AND
```

```
        (:IF
         (<> 0.0
             (diffusion-distance $duration $temperature
               (@ (concentration (above $l)) (spi))
               (@ (concentration $l) (spi))))
         (change - (top-pos $l)
           (diffusion-distance $duration $temperature
             (@ (concentration (above $l)) (spi))
             (@ (concentration $l) (spi)))))
        (:IF
         (<> 0.0
             (diffusion-distance $duration $temperature
               (@ (concentration (below $l)) (spi))
               (@ (concentration $l) (spi))))
         (change + (bottom-pos $l)
           (diffusion-distance $duration $temperature
             (@ (concentration (below $l)) (spi))
             (@ (concentration $l) (spi)))))
        (:IF
         (:OR (> (@ (concentration $l) (spi))
                 (@ (concentration (above $l)) (spi)))
              (> (@ (concentration $l) (spi))
                 (@ (concentration (below $l)) (spi))))
         (change - (concentration $l)
           (+ (diffusion-depletion $duration
                $temperature
                (@ (concentration $l) (spi))
                (@ (concentration (above $l)) (spi)))
              (diffusion-depletion $duration
                $temperature
                (@ (concentration $l) (spi))
                (@ (concentration (below $l)) (spi))))))
        ))))))))
```

*;; Handle changes in layer thickness for non-oxide layers*

```
(for-all-existing $l |:| layer-region (epi)
 (:IF
  (:NOT (:OR (is-type oxide (@ (material $l) (spi)))
             (:AND (n= (@ (top-pos $l) (spi))
                       (@ (top-pos $l) (epi)))
                   (n= (@ (bottom-pos $l) (spi))
                       (@ (bottom-pos $l) (epi))))))
  (change = (thickness $l)
    (- (@ (top-pos $l) (epi))
       (@ (bottom-pos $l) (epi))))))
```

*;; Handle effects on the Oxide layer*

```
(for-all-existing $l \: layer-region (spi)
 (:IF
  (:AND
   (:NOT (is-type oxide (@ (material $l) (spi))))
   (uppermost-oxidation-survivor $l (process interval))
   (is-type oxide
     (@ (material (oxide-layer-region $l)) (epi))))
  (:AND
   (change = (thickness (oxide-layer-region $l))
     (+ (oxide-thickness-above (@ (above $l) (spi)) (spi))
        (* (amount-oxidized $l)
           (oxide-density-ratio (@ (material $l) (spi)))
```

```
            )))
     (change = (bottom-pos (oxide-layer-region $l))
       (- (@ (top-pos $l) (spi))
          (amount-oxidized $l)))
     (change = (top-pos (oxide-layer-region $l))
       (+ (+ (oxide-thickness-above (@ (above $l) (spi))
                 (spi))
             (* (amount-oxidized $l)
                (oxide-density-ratio (@ (material $l)
                                        (spi)))))
                (- (@ (top-pos $l) (spi))
                   (amount-oxidized $l))))
     (change = (above $l) (oxide-layer-region $l))
     (change = (below (oxide-layer-region $l)) $l)))))
  constraints
  ((<= $ox-duration $duration)
   (for-all-existing $l \: layer-region (spi)
    (:=> (:AND (:not (v= (@ (above $l) (spi))
                         THE-ENVIRONMENT))
               (v= (@ (material $l) (spi))
                   (@ (material (above $l)) (epi)))
               (:not (destroyed $l (epi)))
               (v= (@ (above $l) (spi))
                   (@ (above $l) (epi))))
         (:AND (:=> (>  (@ (concentration $l) (spi))
                        (@ (concentration (above $l)) (spi)))
                    (>= (@ (concentration $l) (epi))
                        (@ (concentration (above $l))
                     (epi))))
               (:=> (<  (@ (concentration $l) (spi))
                        (@ (concentration (above $l)) (spi)))
                    (<= (@ (concentration $l) (epi))
                        (@ (concentration (above $l))
                           (epi)))))))
   (for-all-existing $lr |:| layer-region (end-of (PI))
    (:AND
     (> (@ (top-pos $lr) (end-of (process interval)))
        (@ (bottom-pos $lr) (end-of (process interval))))
     (> (@ (thickness $lr) (end-of (process interval)))
        0.0)))
   (for-all-existing $l \: layer-region (epi)
    (:=> (:not (v= (@ (above $l) (epi)) THE-ENVIRONMENT))
         (n= (@ (top-pos $l) (epi))
             (@ (bottom-pos (above $l)) (epi)))))
   (DEFN AMOUNT-OXIDIZED ($l)
     (* (- $ox-duration
           (oxidation-destroy-time
            (@ (above $l) (start-of (process interval)))
            $temperature (start-of (process interval))))
        (oxidation-rate
         $temperature
         (@ (material $l) (start-of (process interval)))
         (oxide-thickness-above
          (@ (above $l) (start-of (process interval)))
          (start-of (process interval))))))
   (DEFN OXIDE-LAYER-REGION ($lr)
     (@ (surface (region-type $lr))
        (end-of (process interval))))
```

```
   (DEFN SPI () (start-of (process interval)))
   (DEFN PI () (process interval))
   (DEFN EPI () (end-of (process interval)))))
```

*;; Simultaneous oxidation and drive-in diffusion. Does not handle enhancement of*
*;; diffusion by oxidation, nor diffusion across material boundaries. Also, does*
*;; not handle loss or pile-up of dopant due to advancing oxide-silicon boundary.*

```
(defprocess DAS-OX/DRIVE-IN
  AKO DAS-Create/Modify-Layers
  parameters (($temperature finite-positive-real)
              ($duration finite-positive-real)
              ($ox-duration finite-positive-real))
  preconds ((> $temperature MINIMUM-DIFFUSION-TEMPERATURE))
  effects
  (;; Layer-regions can be consumed by oxidation or diffusion only
   (for-all-existing $l \: layer-region (spi)
    (:IFF (destroyed $l (epi))
          (:OR (consumed-by-oxidation $l (epi))
               (consumed-by-diffusion $l (epi)))))

   ;; If the surface layer oxidizes, there will be an oxide at the
   ;; surface at the end of the process
   (for-all-existing $rt \: region-type (spi)
    (:=>
     (:AND
      (:NOT (is-type oxide
              (@ (material (surface $rt)) (spi))))
      (> (oxidation-rate $temperature
            (@ (material (surface $rt)) (spi)) 0.0)
         0.0))
     (CREATED
      ($oxide-layer-region layer-region (spi))
      (member $oxide-layer-region
        (new-layer (@ (layer (surface $rt)) (spi)) (PI)))
      (member (@ (layer (surface $rt)) (spi))
        (new-layers (PI)))
      (change = (surface $rt) $oxide-layer-region)
      (change = (region-type $oxide-layer-region) $rt)
      (change = (above $oxide-layer-region) THE-ENVIRONMENT)
      (change = (material $oxide-layer-region) |Oxide|)
      (change = (dopant $oxide-layer-region) *no-dopant*))))

   (for-all-existing $l \: layer-region (spi)
    (:=>
     (:NOT (is-type oxide (@ (material $l) (spi))))
     (:AND
      ;; Oxidation effects
      (:IFF
       (>= $ox-duration
           (oxidation-destroy-time $l $temperature (spi)))
       (consumed-by-oxidation $l (epi)))
      (:IFF
       (:AND
        (>= $ox-duration
            (oxidation-destroy-time (@ (above $l) (spi))
              $temperature (spi)))
```

```
  (:NOT (consumed-by-oxidation $l (epi))))
 (uppermost-oxidation-survivor $l (process interval)))
;; $l is uppermost layer that survives oxidation
(:IF
 (uppermost-oxidation-survivor $l (process interval))
 ;; Oxidation without diffusion (no diffusion between materials)
 (:IF
  (:OR (v= (@ (dopant $l) (spi)) *NO-DOPANT*)
       (:NOT (v= (@ (material $l) (spi))
                 (@ (material (below $l)) (spi)))))
  (:IF
   (:AND
    (> $ox-duration
       (oxidation-destroy-time (@ (above $l) (spi))
         $temperature (spi)))
    (> (oxidation-rate $temperature
         (@ (material $l) (spi))
         (oxide-thickness-above
           (@ (above $l) (spi)) (spi)))
       0.0))
   (change - (top-pos $l) (amount-oxidized $l)))))


(:IF
 (:NOT (consumed-by-oxidation $l (epi)))
 (:AND
  ;; Diffusion effects
  ;; Diffusion from/to the top only--same basic material (no oxidation)
  (:IF
   (:AND
    (:NOT (uppermost-oxidation-survivor $l (PI)))
    (v= (@ (material $l) (spi))
        (@ (material (above $l)) (spi)))
    (:OR (v= (@ (below $l) (spi)) THE-ENVIRONMENT)
         (:NOT (v= (@ (material $l) (spi))
                   (@ (material (below $l)) (spi))))))
   (:AND
    (:IFF (<= (@ (thickness $l) (spi))
              (diffusion-distance $duration
                 $temperature
                 (@ (concentration (above $l)) (spi))
                 (@ (concentration $l) (spi))))
          (consumed-by-diffusion $l (epi)))
    (:IF
     (consumed-by-diffusion $l (epi))
     (:AND
      (change = (below (above $l))
        (@ (below $l) (spi)))
      (:=> (:NOT (V= (@ (below $l) (spi))
                     THE-ENVIRONMENT))
           (change = (above (below $l))
             (@ (above $l) (spi))))))
    (:=>
     (:AND
      (> (@ (thickness $l) (spi))
         (diffusion-distance $duration $temperature
           (@ (concentration (above $l)) (spi))
           (@ (concentration $l) (spi))))
```

```
    (<> 0.0 (diffusion-distance $duration
              $temperature
              (@ (concentration (above $l)) (spi))
              (@ (concentration $l) (spi)))))
   (:AND
    (change - (top-pos $l)
      (diffusion-distance $duration $temperature
        (@ (concentration (above $l)) (spi))
        (@ (concentration $l) (spi))))
    (:IF
     (> (@ (concentration $l) (spi))
        (@ (concentration (above $l)) (spi)))
     (change - (concentration $l)
       (diffusion-depletion $duration $temperature
        (@ (concentration $l) (spi))
        (@ (concentration (above $l)) (spi)))
       ))))))
```

*;; Diffusion from/to the bottom only--*
*;; same basic material (oxidation may be involved)*

```
(:IF
 (:AND
  (:NOT (v= (@ (below $l) (spi)) THE-ENVIRONMENT))
  (v= (@ (material $l) (spi))
      (@ (material (below $l)) (spi)))
  (:OR (v= (@ (above $l) (spi)) THE-ENVIRONMENT)
       (:NOT (v= (@ (material $l) (spi))
                 (@ (material (above $l)) (spi))))
       (uppermost-oxidation-survivor $l (PI))))
 (:AND
  (:IFF (:NOT (uppermost-oxidation-survivor $l (PI)))
        (N= (oxidation-reduced-thickness $l (spi))
            (@ (thickness $l) (spi))))
  (:IFF (uppermost-oxidation-survivor $l (PI))
        (N= (oxidation-reduced-thickness $l (spi))
            (- (@ (thickness $l) (spi))
               (amount-oxidized $l))))
  (:IFF (<= (oxidation-reduced-thickness $l (spi))
            (diffusion-distance $duration
              $temperature
              (@ (concentration (below $l)) (spi))
              (@ (concentration $l) (spi))))
        (consumed-by-diffusion $l (epi)))
  (:IF
   (consumed-by-diffusion $l (epi))
   (:IF
     (:NOT (uppermost-oxidation-survivor $l (PI)))
     (:AND (change = (above (below $l))
             (@ (above $l) (spi)))
           (change = (below (above $l))
             (@ (below $l) (spi))))))
  (:IF
   (:NOT (consumed-by-diffusion $l (epi)))
   (:AND
    (:IF (uppermost-oxidation-survivor $l (PI))
         (change - (top-pos $l)
           (amount-oxidized $l)))
    (:IF
```

```
          (<> 0.0
              (diffusion-distance $duration $temperature
                (@ (concentration (below $l)) (spi))
                (@ (concentration $l) (spi))))
          (:AND
           (change + (bottom-pos $l)
             (diffusion-distance $duration $temperature
               (@ (concentration (below $l)) (spi))
               (@ (concentration $l) (spi))))
           (:IF (> (@ (concentration $l) (spi))
                   (@ (concentration (below $l)) (spi)))
                (change - (concentration $l)
                  (diffusion-depletion $duration
                    $temperature
                    (@ (concentration $l) (spi))
                    (@ (concentration (below $l)) (spi)))
                  ))))))))
```

*;; Diffusion from/to both top and bottom--*
*;; same basic material (no oxidation)*

```
(:IF
 (:AND
  (:NOT (v= (@ (above $l) (spi)) THE-ENVIRONMENT))
  (:NOT (v= (@ (below $l) (spi)) THE-ENVIRONMENT))
  (v= (@ (material $l) (spi))
      (@ (material (below $l)) (spi)))
  (v= (@ (material $l) (spi))
      (@ (material (above $l)) (spi)))
  (:NOT (uppermost-oxidation-survivor $l (PI))))
 (:AND
  (:IFF
   (<= (@ (thickness $l) (spi))
       (+ (diffusion-distance $duration $temperature
            (@ (concentration (above $l)) (spi))
            (@ (concentration $l) (spi)))
          (diffusion-distance $duration $temperature
            (@ (concentration (below $l)) (spi))
            (@ (concentration $l) (spi)))))
   (consumed-by-diffusion $l (epi)))
  (:IF
   (consumed-by-diffusion $l (epi))
   (:AND (change = (above (below $l))
           (@ (above $l) (spi)))
         (change = (below (above $l))
           (@ (below $l) (spi)))))
  (:IF
   (:AND
    (:NOT (consumed-by-diffusion $l (epi)))
    (:OR
     (<> 0.0
         (diffusion-distance $duration $temperature
           (@ (concentration (above $l)) (spi))
           (@ (concentration $l) (spi))))
     (<> 0.0
         (diffusion-distance $duration $temperature
           (@ (concentration (below $l)) (spi))
           (@ (concentration $l) (spi))))))
   (:AND
```

```
         (:IF
          (<> 0.0
              (diffusion-distance $duration $temperature
                (@ (concentration (above $l)) (spi))
                (@ (concentration $l) (spi))))
          (change - (top-pos $l)
                  (diffusion-distance $duration
                    $temperature
                    (@ (concentration (above $l)) (spi))
                    (@ (concentration $l) (spi)))))
         (:IF
          (<> 0.0
              (diffusion-distance $duration $temperature
                (@ (concentration (below $l)) (spi))
                (@ (concentration $l) (spi))))
          (change + (bottom-pos $l)
            (diffusion-distance $duration $temperature
             (@ (concentration (below $l)) (spi))
             (@ (concentration $l) (spi)))))
         (:IF
          (:OR (> (@ (concentration $l) (spi))
                  (@ (concentration (above $l)) (spi)))
               (> (@ (concentration $l) (spi))
                  (@ (concentration (below $l)) (spi))))
          (change - (concentration $l)
            (+ (diffusion-depletion $duration $temperature
                (@ (concentration $l) (spi))
                (@ (concentration (above $l)) (spi)))
               (diffusion-depletion $duration $temperature
                (@ (concentration $l) (spi))
                (@ (concentration (below $l)) (spi))))))
         )))))))))
;; Handle changes in layer thickness for non-oxide layers
(for-all-existing $l |:| layer-region (epi)
 (:IF
  (:NOT
   (:OR (is-type oxide (@ (material $l) (spi)))
        (:AND (n= (@ (top-pos $l) (spi))
                  (@ (top-pos $l) (epi)))
              (n= (@ (bottom-pos $l) (spi))
                  (@ (bottom-pos $l) (epi))))))
  (change = (thickness $l)
    (- (@ (top-pos $l) (epi))
       (@ (bottom-pos $l) (epi))))))
;; Handle effects on the Oxide layer
(for-all-existing $l \: layer-region (spi)
 (:IF
  (:AND
   (:NOT (is-type oxide (@ (material $l) (spi))))
   (uppermost-oxidation-survivor $l (process interval))
   (is-type oxide
     (@ (material (oxide-layer-region $l)) (epi))))
  (:AND
   (change = (thickness (oxide-layer-region $l))
     (+ (oxide-thickness-above (@ (above $l) (spi)) (spi))
        (* (amount-oxidized $l)
           (oxide-density-ratio (@ (material $l) (spi))))))
   (change = (bottom-pos (oxide-layer-region $l))
```

```
            (- (@ (top-pos $l) (spi))
               (amount-oxidized $l)))
    (change = (top-pos (oxide-layer-region $l))
      (+ (+ (oxide-thickness-above
              (@ (above $l) (spi)) (spi))
            (* (amount-oxidized $l)
               (oxide-density-ratio
                 (@ (material $l) (spi)))))
         (- (@ (top-pos $l) (spi))
            (amount-oxidized $l))))
    (:IF (destroyed $l (epi))
         (:AND
          (change = (above (below $l))
            (oxide-layer-region $l))
          (change = (below (oxide-layer-region $l))
            (@ (below $l) (spi)))))
    (:IF (:NOT (destroyed $l (epi)))
         (:AND
          (change = (above $l) (oxide-layer-region $l))
          (change = (below (oxide-layer-region $l))
            $l)))))))
constraints
((<= $ox-duration $duration)
 (for-all-existing $l \: layer-region (spi)
  (:=>
   (:AND (:not (v= (@ (above $l) (spi))
                   THE-ENVIRONMENT))
         (v= (@ (material $l) (spi))
             (@ (material (above $l)) (epi)))
         (:not (destroyed $l (epi)))
         (v= (@ (above $l) (spi)) (@ (above $l) (epi))))
   (:AND (:=> (>  (@ (concentration $l) (spi))
                  (@ (concentration (above $l)) (spi)))
              (>= (@ (concentration $l) (epi))
                  (@ (concentration (above $l)) (epi))))
         (:=> (<  (@ (concentration $l) (spi))
                  (@ (concentration (above $l)) (spi)))
              (<= (@ (concentration $l) (epi))
                  (@ (concentration (above $l)) (epi)))))))
 (for-all-existing $lr |:| layer-region (EPI)
  (:AND
   (> (@ (top-pos $lr) (end-of (process interval)))
      (@ (bottom-pos $lr) (end-of (process interval))))
   (> (@ (thickness $lr) (end-of (process interval)))
      0.0)))

 (for-all-existing $l \: layer-region (epi)
  (:=> (:not (v= (@ (above $l) (epi)) THE-ENVIRONMENT))
       (n= (@ (top-pos $l) (epi))
           (@ (bottom-pos (above $l)) (epi)))))
 (DEFN AMOUNT-OXIDIZED ($l)
   (* (- $ox-duration
         (oxidation-destroy-time
          (@ (above $l) (start-of (process interval)))
          $temperature (start-of (process interval))))
      (oxidation-rate
       $temperature
       (@ (material $l) (start-of (process interval)))
```

```
        (oxide-thickness-above
         (@ (above $l) (start-of (process interval)))
         (start-of (process interval))))))
  (DEFN OXIDE-LAYER-REGION ($lr)
    (@ (surface (region-type $lr))
       (end-of (process interval))))
  (DEFN PI () (process interval))
  (DEFN SPI () (start-of (process interval)))
  (DEFN EPI () (end-of (process interval)))))
```

*;; Diffusion from polysilicon into silicon, creating a new doped region*
*;; Highly specific to the Stanford BiCMOS process---used to create emitter region*
*;; and N+ Source/Drain contacts.*

```
(defprocess DAS-Poly-Source-Diffusion
  AKO DAS-Ox/Drive-in
  effects
  ((for-all-existing $l \: layer-region (SPI)
     (:IF
      (:AND
       (:NOT (consumed-by-oxidation $l (EPI)))
       (is-type polysilicon (@ (material $l) (SPI)))
       (is-type silicon (@ (material (below $l)) (SPI)))
       (:NOT (is-type polysilicon
               (@ (material (below $l)) (SPI)))))
      (:IF
       (> (@ (concentration $l) (SPI))
          (@ (concentration (below $l)) (SPI)))
       (CREATED ($NL layer-region (SPI))
        (member $NL (new-layer
                      (@ (layer (below $l)) (SPI)) (PI)))
        (member (@ (layer (below $l)) (SPI))
          (new-layers (process interval)))
        (change = (below $l) $nl)
        (change = (region-type $nl)
          (@ (region-type $l) (SPI)))
        (change = (top-pos $nl)
          (@ (top-pos (below $l)) (SPI)))
        (change = (above $nl) $l)
        (change - (concentration $l)
          (/ (xfer-dose $l) (@ (thickness $l) (SPI))))
        (change - (top-pos (below $l)) (junction-depth $l))
        (change - (thickness (below $l)) (junction-depth $l))
        (change = (bottom-pos $nl)
          (- (@ (top-pos (below $l)) (SPI))
             (junction-depth $l)))
        (change = (thickness $nl) (junction-depth $l))
        (change = (below $nl) (@ (below $l) (SPI)))
        (change = (dopant $nl) (@ (dopant $l) (SPI)))
        (:IF (v= (@ (dopant-type (dopant $l)) (SPI))
                 (@ (dopant-type (dopant (below $l))) (SPI)))
             (change = (concentration $nl)
                      (@ (concentration $l) (EPI))))
        (:IF
         (:NOT
          (v= (@ (dopant-type (dopant $l)) (SPI))
              (@ (dopant-type (dopant (below $l))) (SPI))))
         (change = (concentration $nl)
                 (- (@ (concentration $l) (EPI))
```

```
                  (@ (concentration (below $l)) (SPI)))))
      (change = (material $nl)
        (@ (material (below $l)) (SPI)))
      (change = (above (below $l)) $nl)))
    )))
constraints
((DEFN XFER-DOSE ($l)
   (transferred-dose $duration $temperature $l
     (@ (below $l) (SPI))))
 (DEFN Junction-Depth ($l)
   (Diffused-Junction-depth $duration $temperature $l
     (@ (below $l) (SPI)))))
)
```

# Appendix C: Axioms Supporting the Models

This appendix exhibits the generic predicate and generic functions, as well as a few "daemons" employed by the DAS qualitative simulator.

```
(defnoticer Is-Type ((IS-TYPE ?type ?object) :intern Gordius::*process*)
  (unless (rup:free-var ^(IS-TYPE ?type ?object))
    (let ((class (rup::term-name ^?type))
          (instance (history::term-teval ^?object)))
      (unless (unknown? instance)
        (assert-expr ^(IS-TYPE ?type ?object) 'DOMAIN-MODEL
                     (if (typep instance class) :TRUE :FALSE))))))

(def-generic-function Any-Member ((set (set . temporal-object)))
                                  temporal-object
  (when (not (null (history::set-members set)))
    (gd::set-term-value Any-Member (first (history::set-members set)))))

(def-generic-predicate Is-Soft-Photoresist? ((object temporal-object))
  (assert-expr ^?Is-Soft-Photoresist? 'DOMAIN-MODEL
               (if (typep object '(and
                                   (or exposed-pos-photoresist
                                       exposed-photoresist
                                       neg-photoresist)
                                   (not exposed-neg-photoresist)))
                   :TRUE :FALSE)))

(def-generic-predicate Destroyed ((object temporal-object) (time time)))
(def-generic-predicate Converted ((object temporal-object) (time time)))
(def-generic-predicate Consumed-by-Etching
   ((object temporal-object) (time time)))
(def-generic-predicate Consumed-by-Oxidation
   ((object temporal-object) (time time)))
(def-generic-predicate Consumed-by-Diffusion
   ((object temporal-object) (time time)))
(def-generic-predicate Uppermost-Oxidation-Survivor
   ((l layer-region) (i time-interval)))

(def-generic-predicate Adherent
      ((super-mat material-type) (sub-mat material-type))
  (assert-expr ^?Adherent 'DOMAIN-MODEL
    (if (cond ((typep super-mat 'polysilicon)
               (not (typep sub-mat 'photoresist)))
              ((typep super-mat 'nitride)
               (or (typep sub-mat 'oxide) (typep sub-mat 'polysilicon)))
              ((typep super-mat 'photoresist)
               (not (typep sub-mat 'photoresist)))
              ((typep super-mat 'tungsten) (typep sub-mat 'polysilicon))
```

```
              ((typep super-mat 'oxide)
               (not (typep sub-mat 'photoresist)))
              ((typep super-mat 'metal)
               (not (typep sub-mat 'photoresist)))
              ((typep super-mat 'silicon) (typep sub-mat 'silicon)))
        :TRUE :FALSE)))

(def-generic-predicate Accepts-Dopant?
       ((mat material-type) (dop dopant))
  ;; For now, allow any material to accept any dopant
  (assert-expr ^?Accepts-Dopant? 'DOMAIN-MODEL :TRUE))

(def-generic-function New-Layer
    ((old-layer layer) (step time-interval)) (set . layer-region))

(def-generic-function Etch-Destroy-Time
    ((l layer-region) (i time-interval)) real
  (if (typep l 'ENVIRONMENT)
      (assert-expr ^(n= ?Etch-Destroy-Time 0.0) 'DOMAIN-MODEL)
      (let ((layer (rup:seal l))
           (i-term (second (rup::subterms (rup:seal (start-of i))))))
        (assert-expr
         ^(n= ?Etch-Destroy-Time
              (+ (etch-destroy-time (@ (above ?layer)
                                       (start-of ?i-term)) ?i-term)
                 (/ (@ (thickness ?layer) (start-of ?i-term))
                    (etch-rate ?layer ?i-term))))
          'DOMAIN-MODEL))))


;;; All etchants have finite, non-negative etch-rates for all materials
(defnoticer All-Etch-Rates-Finite
   ((PARAMETER-OF ?process $ETCHANT ?etchant) :TRUE GD::*process*)
  (let ((I (process::get-time-of-process ^?process)))
    (assert-expr
     ^(FOR-ALL-EXISTING $L \: Layer-Region (start-of ?I)
        (:AND
         (<= 0.0 (etch-rate $L ?I))
         (<  (etch-rate $L ?I) +INFINITY)))
     'DOMAIN-MODEL)))

(eval-when (load eval compile)
  (defmacro defEtchRateRule (etchant (layer-region interval) rule-form)
    `(defnoticer ,(intern (concatenate 'string
                           (symbol-name etchant) "-ETCH-RATE"))
         ((PARAMETER-OF ?process $ETCHANT ,etchant) :TRUE GD::*process*)
       (let ((I (process::get-time-of-process ^?process)))
         (assert-expr
          ,(rup::term-expand
            (rup::parse-term
            `(FOR-ALL-EXISTING ,layer-region \: Layer-Region
                               (start-of ?I)
               ,(subst '?I interval rule-form))))
          'DOMAIN-MODEL)))))
```

```
;;; |6:1 Buffered HF| etches Oxide, but not silicon or nitride?
(defEtchRateRule |6:1 Buffered HF| ($L $I)
  (:AND
   (:=> (is-type oxide        (@ (material $L) (start-of $I)))
        (> (etch-rate $L $I) 0.0))
   (:=> (:OR
         (is-type nitride     (@ (material $L) (start-of $I)))
         (is-type silicon     (@ (material $L) (start-of $I)))
         (is-type photoresist (@ (material $L) (start-of $I))))
        (N= (etch-rate $L $I) 0.0))))

;;; |50:1 HF| etches Oxide, but not silicon or nitride?
(defEtchRateRule |50:1 HF| ($L $I)
  (:AND
   (:=> (is-type oxide        (@ (material $L) (start-of $I)))
        (> (etch-rate $L $I) 0.0))
   (:=> (:OR
         (is-type nitride     (@ (material $L) (start-of $I)))
         (is-type silicon     (@ (material $L) (start-of $I)))
         (is-type photoresist (@ (material $L) (start-of $I))))
        (N= (etch-rate $L $I) 0.0))))

;;; Refluxed-H3PO4 etches oxide and nitride but not silicon?
(defEtchRateRule Refluxed-H3PO4 ($L $I)
  (:AND
   (:=> (:OR
         (is-type nitride     (@ (material $L) (start-of $I)))
         (is-type oxide       (@ (material $L) (start-of $I))))
        (> (etch-rate $L $I) 0.0))
   (:=> (:OR
         (is-type silicon     (@ (material $L) (start-of $I)))
         (is-type photoresist (@ (material $L) (start-of $I))))
        (N= (etch-rate $L $I) 0.0))))

(def-generic-function Layer-Closure ((layer layer) (time time))
                                    (set . layer-region)
  (assert-expr
   ^(v= ?Layer-Closure
        ?(rup::termhashcons2
          (cons ^set
                (mapcar #'rup::seal
                        (history::set-members
                         (attribute-at-time
                            layer 'layer-regions time))))))
    'Closed-World-Assumption))

(def-generic-function Detected-Layer-Regions
    ((step time-interval)) (set . layer-region))

(def-generic-function Solid-Solubility
     ((mat material-type) (dopant dopant)) positive-real
  (assert-expr ^(> ?Solid-Solubility ++CONCENTRATION) 'DOMAIN-MODEL))
```

```
(defnoticer Pre-Dep-Conc ((PRE-DEP-CONC ?vapor-pressure ?mat ?dopant)
                          :intern Gordius::*process*)
  (unless (rup:free-var ^(PRE-DEP-CONC ?vapor-pressure ?mat ?dopant))
    (let ((solid-solubility ^(solid-solubility ?mat ?dopant)))
      (assert-expr
       ^(:and (<= (PRE-DEP-CONC ?vapor-pressure ?mat ?dopant)
                  ?solid-solubility)
              (:=> (>= ?vapor-pressure PRE-DEP-THRESHOLD)
                   (n= (PRE-DEP-CONC ?vapor-pressure ?mat ?dopant)
                       ?solid-solubility)))
        'DOMAIN-MODEL))))
(rup:init-form (assert-expr '(FUNCTIONAL PRE-DEP-CONC
                              (((>= ARG1 PRE-DEP-THRESHOLD) IND)
                               ((< ARG1 PRE-DEP-THRESHOLD) MI))
                              NA NA)))

;;; Generic functions and predicates for photolithography model.
(def-generic-function EXPOSED-REGION-TYPE
     ((ld region-type) (i time-interval)) region-type)
(def-generic-function EXPOSED-LAYER-REGION
     ((l layer-region) (i time-interval)) layer-region)
(def-generic-function ALTERED-REGIONS
     ((ld region-type) (i time-interval)) (set . wafer-region))
(def-generic-function UNALTERED-REGIONS
     ((ld region-type) (i time-interval)) (set . wafer-region))
(def-generic-function UNALTERED-REGION-TYPES
     ((i time-interval)) (set . region-type))
(def-generic-function UNALTERED-REGION-TYPES-CLOSURE
     ((i time-interval)) (set . region-type)
  (let* ((i-term (second (rup::subterms (rup:seal (start-of i)))))
         (set (gd::term-teval ^(UNALTERED-REGION-TYPES ?i-term))))
    (if (history::unknown-object-p set)
        (assert-expr
         ^(:NOT (exists $rt e ?Unaltered-Region-Types-Closure))
         'Closed-World-Assumption)
        (assert-expr
         ^(v= ?Unaltered-Region-Types-Closure
              ?(rup::termhashcons2
                (cons ^set
                      (mapcar #'rup::seal (history::set-members set)))))
          'Closed-World-Assumption))))

(proclaim '(special neg-photoresist-exposed pos-photoresist-exposed
                    photoresist-exposed))

(def-generic-function EXPOSED-MATERIAL
     ((mat material-type)) material-type
  (gd::set-term-value
    exposed-material (exposed-material mat) 'DOMAIN-MODEL))

(gd::init-form
  (assert-expr '(OLD-MASK-REGION-TYPE LD1 *BOTTOM*) 'DOMAIN-MODEL))

(defnoticer Old-Mask-Region-Type
     ((EXPOSED-REGION-TYPE ?ignore ?ignore) :INTERN GD::*PROCESS*)
  (unless (gd::free-var ^(EXPOSED-REGION-TYPE ?ignore ?ignore))
    (gd::add-value-noticer gd::*process*
      #'old-mask-region-type ^(EXPOSED-REGION-TYPE ?ignore ?ignore))))
```

```
(defun old-mask-region-type (term)
  (when (gd::real-object? (gd::term-value term))
    (assert-expr
     ^(OLD-MASK-REGION-TYPE
       ?(gd::seal (gd::term-value term))
       ?(gd::seal (gd::term-value (second (gd::subterms term)))))
    'DOMAIN-MODEL)))

(def-generic-function DIFFUSION-DISTANCE
    (($duration positive-real) ($temperature positive-real)
     ($concentration1 real) ($concentration2 real))
    REAL
  (assert-expr
    ^(?(gd::term (gd::qrelations $concentration1 $concentration2))
       ?Diffusion-Distance 0.0)
    'DOMAIN-MODEL)
  (let* ((args (cdr (gd::subterms Diffusion-Distance)))
         (duration (first args))
         (temperature (second args))
         (c1 (third args))
         (c2 (fourth args)))
    (assert-expr
     ^(n= ?Diffusion-Distance
          (minus (diffusion-distance
                   ?duration ?temperature ?c2 ?c1))))))

(gd::init-form
  (assert-expr
   '(FUNCTIONAL DIFFUSION-DISTANCE
      (((N= ARG3 ARG4) IND) ((< ARG3 ARG4) MD) ((> ARG3 ARG4) MI))
      (((N= ARG3 ARG4) IND) ((< ARG3 ARG4) MD) ((> ARG3 ARG4) MI))
      MI MD)))

(def-generic-function DIFFUSION-DEPLETION
     (($duration positive-real) ($temperature positive-real)
      ($encroacher-concentration real) ($encroachee-concentration real))
     REAL
  (cond ((gd::qis-true? $encroacher-concentration '>
                        $encroachee-concentration)
         (assert-expr ^(> ?Diffusion-Depletion 0.0) 'DOMAIN-MODEL)
         (assert-expr ^(<= ?Diffusion-Depletion
                           (- ?(gd::seal $encroacher-concentration)
                              ?(gd::seal $encroachee-concentration)))
                 'DOMAIN-MODEL))
        (t (assert-expr ^(n= ?Diffusion-Depletion 0.0) 'DOMAIN-MODEL))))

(gd::init-form
 (assert-expr '(FUNCTIONAL DIFFUSION-DEPLETION MI MI MI MD)))

(def-generic-function Transmission-Energy
     ((l layer-region) (time time)) non-negative-real
  (let ((layer (gd::seal l))
        (time-term (gd::seal time)))
    (cond ((typep l 'environment)
           (assert-expr ^(n= ?Transmission-Energy 0.0) 'DOMAIN-MODEL))
          ((typep (history::@ (history::oget-safe l 'material) time)
                  'photoresist)
```

```
           ;; Assume implants never pass through photoresist
           (assert-expr
            ^(n= ?Transmission-Energy +INFINITY)
            'DOMAIN-MODEL))
          (T
           (assert-expr
            ^(n= ?Transmission-Energy
                 (+ (transmission-energy
                      (@ (above ?layer) ?time-term) ?time-term)
                    (/ (@ (thickness ?layer) ?time-term)
                       (energy-loss-rate
                         (@ (material ?layer) ?time-term)))))
            'DOMAIN-MODEL)))))

(def-generic-function Energy-Loss-Rate
     ((mat material-type)) positive-real)

(def-generic-function Ion-Spread
     ((median-distance positive-real)) positive-real)
(gd::init-form (assert-expr '(FUNCTIONAL ION-SPREAD MI)))

(def-generic-function Implanted-Conc
     ((r region-type) (i time-interval)) positive-real
  (let* ((i-term (second (rup::subterms (rup:seal (start-of i)))))
         (region-term (gd::seal r))
         (dose-term (process::get-process-parameter
                     ^$dose (process::get-process-at i-term))))
    (assert-expr
     ^(n= ?implanted-Conc
          (ion-conc ?dose-term
                    (ion-spread (median-implant-distance
                                 ?region-term ?i-term)))))))

(def-generic-function Ion-Conc
      ((dose positive-real) (spread positive-real)) positive-real
  (let ((dose-term (gd::seal dose))
        (spread-term (gd::seal spread)))
    (assert-expr ^(n= ?ion-conc (/ ?dose-term ?spread-term)))))
(gd::init-form (assert-expr '(FUNCTIONAL ION-CONC MI MD)))

(def-generic-function Ion-Top ((rt region-type) (I time-interval)) real
  (let ((r-term (gd::seal rt))
        (i-term (second (rup::subterms (rup:seal (start-of I))))))
    (assert-expr
     ^(N= ?Ion-Top
          (+ (median-implant-position ?r-term ?i-term)
             (ion-spread (median-implant-distance ?r-term ?i-term)))))
    (assert-expr ^(> ?Ion-Top (ion-bottom ?r-term ?i-term)))))

(def-generic-function Ion-Bottom
      ((rt region-type) (I time-interval)) real
  (let ((r-term (gd::seal rt))
        (i-term (second (rup::subterms (rup:seal (start-of I))))))
    (assert-expr
     ^(N= ?Ion-Bottom
          (- (median-implant-position ?r-term ?i-term)
             (ion-spread (median-implant-distance ?r-term ?i-term)))))
    (assert-expr ^(> (ion-top ?r-term ?i-term) ?Ion-Bottom))))
```

```
(def-generic-function Median-Implant-Distance
     ((rt region-type) (I time-interval)) positive-real)

(def-generic-function Median-Implant-Position
     ((rt region-type) (I time-interval)) real
  (let* ((r-term (gd::seal rt))
         (t-term (rup:seal (start-of I)))
         (i-term (second (rup::subterms t-term))))
    (assert-expr ^(N= ?Median-Implant-Position
                      (- (@ (top-pos (surface ?r-term)) ?t-term)
                         (median-implant-distance ?r-term ?i-term))))))

(def-generic-function IMPLANTED-LAYER-REGION
     ((l layer-region) (i time-interval)) layer-region)

(def-generic-function Oxidation-Rate
     ((temperature positive-real) (mat material-type)
      (oxide-thickness-above positive-real)) finite-non-negative-real
  (cond ((typep mat 'silicon)
         (assert-expr
          ^(n= ?Oxidation-rate silicon-oxidation-rate)))
        ((typep mat 'nitride)
         (assert-expr
          ^(n= ?Oxidation-rate silicon-nitride-oxidation-rate)))))
(gd::init-form (assert-expr '(> silicon-oxidation-rate 0.0)))
(gd::init-form (assert-expr '(> silicon-nitride-oxidation-rate 0.0)))
(gd::init-form
  (assert-expr
   '(> silicon-oxidation-rate silicon-nitride-oxidation-rate)))
(gd::init-form (assert-expr '(FUNCTIONAL OXIDATION-RATE MI NA MD)))

(def-generic-function Oxidation-Destroy-Time
    ((l layer-region) (temperature positive-real) (time time))
    non-negative-real
  (let ((layer (gd::seal l))
        (temperature-term (gd::seal temperature))
        (time-term (gd::seal time)))
    (cond
     ((typep l 'environment)
      (assert-expr ^(n= ?Oxidation-Destroy-Time 0.0)))
     ((typep (history::@ (history::oget-safe l 'material) time) 'oxide)
      (assert-expr
       ^(n= ?Oxidation-Destroy-Time
            (oxidation-destroy-time
              (@ (above ?layer) ?time-term)
              ?temperature-term ?time-term))))
     (T
      (assert-expr
       ^(n= ?Oxidation-Destroy-Time
            (+ (oxidation-destroy-time (@ (above ?layer) ?time-term)
                                       ?temperature-term ?time-term)
               (/ (@ (thickness ?layer) ?time-term)
                  (oxidation-rate
                   ?temperature-term (@ (material ?layer) ?time-term)
                   (oxide-thickness-above
                    (@ (above ?layer) ?time-term) ?time-term)))))))))))
```

```
(def-generic-function Oxide-Density-Ratio
      ((mat material-type)) finite-positive-real
  (cond ((typep mat 'silicon)
         (assert-expr
          ^(n= ?Oxide-Density-Ratio silicon-oxide-density-ratio)))
        ((typep mat 'nitride)
         (assert-expr
          ^(n= ?Oxide-Density-Ratio
               silicon-nitride-oxide-density-ratio)))))
(gd::init-form
 (assert-expr
  '(n= silicon-oxide-density-ratio #.(/ 1 0.44)) 'DOMAIN-MODEL))
(gd::init-form
  `(assert-expr '(> silicon-nitride-oxide-density-ratio 0.0)
                'DOMAIN-MODEL))
(gd::init-form
 (assert-expr
  '(> silicon-oxide-density-ratio silicon-nitride-oxide-density-ratio)
  'DOMAIN-MODEL))

(def-generic-function Oxide-Thickness-Above
      ((l layer-region) (time time)) finite-non-negative-real
  (let ((layer (gd::seal l))
        (time-term (gd::seal time)))
    (cond ((typep l 'environment)
           (assert-expr ^(n= ?Oxide-Thickness-Above 0.0) 'DOMAIN-MODEL))
          ((typep (attribute-at-time l 'material time) 'oxide)
           (assert-expr
            ^(n= ?Oxide-Thickness-Above
                 (+ (@ (thickness ?layer) ?time-term)
                    (oxide-thickness-above
                      (@ (above ?layer) ?time-term) ?time-term)))))
          (t
           (assert-expr
            ^(N= ?Oxide-Thickness-Above
                 (+ (oxide-thickness-above
                      (@ (above ?layer) ?time-term) ?time-term)
                    (* (@ (thickness ?layer) ?time-term)
                       (oxide-density-ratio
                         (@ (material ?layer) ?time-term))))))))))

(def-generic-function Resistance
      ((l layer-region) (fr wafer-region) (time time)
       (orientation (one-of (constant :horizontal)
                            (constant :vertical))))
      positive-real
  (let ((layer (gd::seal l)) (wafer-region (gd::seal fr))
        (time-term (gd::seal time)))
    (assert-expr
     ^(N= ?Resistance
          (* (resistivity ?layer ?time-term)
             ?(case orientation
                (:HORIZONTAL ^(/ (width ?wafer-region ?time-term)
                                 (* (@ (thickness ?layer) ?time-term)
                                    (depth ?layer ?time-term))))
                (:VERTICAL ^(/ (@ (thickness ?layer) ?time-term)
                               (* (width ?wafer-region ?time-term)
                                  (depth ?layer ?time-term)))))))))
```

```
(def-generic-function Resistivity
      ((l layer-region) (time time)) finite-positive-real
  (let* ((layer (gd::seal l)) (time-term (gd::seal time))
         (dopant ^(@ (dopant ?layer) ?time-term))
         (concentration ^(@ (concentration ?layer) ?time-term))
         (sigma (cond ((typep (term-teval dopant) 'N-DOPANT)
                       ^(+ (* (electron-mobility ?concentration)
                              ?concentration)
                           (* (hole-mobility ?concentration)
                              (/ intrinsic-*-squared ?concentration))))
                      (T ^(+ (* (hole-mobility ?concentration)
                                ?concentration)
                             (* (electron-mobility ?concentration)
                                (/ intrinsic-*-squared
                                   ?concentration)))))))
    (assert-expr ^(N= ?Resistivity (/ 1.0 (* Q ?sigma))))))

(def-generic-function Hole-Mobility
     ((concentration real)) finite-positive-real)
(def-generic-function Electron-Mobility
     ((concentration real)) finite-positive-real)

(def-generic-function Width
   ((wr wafer-region) (time time)) finite-positive-real)
(def-generic-function Depth
   ((l layer-region) (time time)) finite-positive-real)

(gd::init-form
 (progn (make-objectq Q 'finite-positive-real :CONSTANT-QUANTITY-P T)
        (make-objectq intrinsic-*-squared 'finite-positive-real
                      :CONSTANT-QUANTITY-P T)))
```

# Appendix D: AESOP Causal Associations

This appendix lists in an abbreviated form the causal associations between the Process-Anomaly level and the Physical-Structure-Anomaly level in the AESOP knowledge base. The first column of the table contains a number classifying the type of the association with respect to our theoretical models. Associations labelled with a 1 treat phenomena that lie with the Phenomena Scope of the models. Associations labelled with a 2 also treat phenomena "understood" by the models, but that involve topological changes to either the process plan or the wafer structure . Causal associations labelled with a 3 treat phenomena that fall outside the competence of the theoretical models.

The remaining columns in the table express the association in the causal direction. The second column is the name of a Process-Anomaly. The column after the right-pointing arrow names a Physical-Structure-Anomaly that might be caused by the Process-Anomaly. The final column lists the qualitative estimate of the likelyhood that occurence of the indicated Process-Anomaly will manifest the Physical-Structure-Anomaly.

For example, the 2nd category causal association in the table indicates that Gate-Ox-Not-Done (omission of the step that grows the gate oxide layer) necessarily causes the condition Gate-Oxide-Non-Existent .

**Glossary**

Gate-Ox[ide]: The thin layer of silicon dioxide that forms the dielectric insulator between the gate and the channel of a MOS transistor;
N[P]-Vt-II: The implanted dose of the NMOS[PMOS] threshold adjust implantation step;
Poly-Gate-Under[Over]-Etch: insufficient[excessive] etching during the etch step that patterns the polysilicon layer used as the gate in the MOS transistors.
Sub-Start-Conc: The concentration of dopant in the initial wafer substrate.
Well-II: Theimplanted dose of the N-Well implantation step.
N[P]-Ch-Conc: The concentration of dopant in the channel of the NMOS[PMOS] device;
Channel-Length: The distance between the source and drain of a MOS device.
Sub[Well]-Conc: The concentration of dopant in the substrate[N-Well].

## Causal Associations

| | Cause | | Effect | Likelihood |
|---|---|---|---|---|
| 1 | Gate-Ox-Temp-High | → | Gate-Oxide-Thick | Maybe |
| 1 | Gate-Ox-Temp-Low | → | Gate-Oxide-Thin | Maybe |
| 1 | Gate-Ox-Temp-Very-High | → | Gate-Oxide-Very-Thick | Maybe |
| 1 | Gate-Ox-Temp-Very-Low | → | Gate-Oxide-Very-Thin | Maybe |
| 1 | Gate-Ox-Time-Long | → | Gate-Oxide-Thick | Very-Likely |
| 1 | Gate-Ox-Time-Short | → | Gate-Oxide-Thin | Very-Likely |
| 1 | Gate-Ox-Time-Very-Long | → | Gate-Oxide-Very-Thick | Very-Likely |
| 1 | Gate-Ox-Time-Very-Short | → | Gate-Oxide-Very-Thin | Very-Likely |
| 1 | N-Vt-II-High | → | N-Ch-Conc-High | Must |
| 1 | N-Vt-II-Low | → | N-Ch-Conc-Low | Must |
| 1 | N-Vt-II-Very-High | → | N-Ch-Conc-Very-High | Must |
| 1 | N-Vt-II-Very-Low | → | N-Ch-Conc-Very-Low | Must |
| 1 | P-Vt-II-High | → | N-Ch-Conc-High | Very-Likely |
| 1 | P-Vt-II-High | → | P-Ch-Conc-Low | Likely |
| 1 | P-Vt-II-Low | → | N-Ch-Conc-Low | Very-Likely |
| 1 | P-Vt-II-Low | → | P-Ch-Conc-High | Very-Likely |
| 1 | P-Vt-II-Very-High | → | N-Ch-Conc-Very-High | Very-Likely |
| 1 | P-Vt-II-Very-High | → | P-Ch-Conc-Very-Low | Very-Likely |
| 1 | P-Vt-II-Very-Low | → | N-Ch-Conc-Very-Low | Very-Likely |
| 1 | P-Vt-II-Very-Low | → | P-Ch-Conc-Very-High | Likely |
| 1 | Sub-Start-Conc-High | → | P-Ch-Conc-Low | Very-Likely |
| 1 | Sub-Start-Conc-High | → | Sub-Conc-High | Very-Likely |
| 1 | Sub-Start-Conc-Low | → | Sub-Conc-Low | Very-Likely |
| 1 | Sub-Start-Conc-Very-High | → | P-Ch-Conc-Low | Very-Likely |
| 1 | Sub-Start-Conc-Very-High | → | Sub-Conc-Very-High | Very-Likely |
| 1 | Sub-Start-Conc-Very-Low | → | Sub-Conc-Very-Low | Very-Likely |
| 1 | Well-II-High | → | P-Ch-Conc-High | Likely |
| 1 | Well-II-High | → | Well-Conc-High | Very-Likely |
| 1 | Well-II-Low | → | P-Ch-Conc-Low | Very-Likely |
| 1 | Well-II-Low | → | Well-Conc-Low | Very-Likely |
| 1 | Well-II-Very-High | → | P-Ch-Conc-Very-High | Very-Likely |
| 1 | Well-II-Very-High | → | Well-Conc-Very-High | Very-Likely |
| 1 | Well-II-Very-Low | → | P-Ch-Conc-Very-Low | Likely |
| 1 | Well-II-Very-Low | → | Well-Conc-Very-Low | Very-Likely |
| 2 | Gate-Ox-Not-Done | → | Gate-Oxide-Non-Existent | Must |
| 2 | N-Vt-II-Not-Masked | → | P-Ch-Conc-High | Maybe |
| 2 | Sub-Start-Conc-Wrong-Type | → | Sub-Conc-Wrong-Type | Very-Likely |
| 2 | Well-II-Not-Done | → | Well-Non-Existent | Very-Likely |
| 2 | Well-II-Not-Masked | → | Sub-Conc-Wrong-Type | Maybe |
| 2 | Well-II-Wrong-Species | → | Well-Non-Existent | Maybe |
| 3 | Gate-Ox-Time-Long | → | P-Ch-Conc-High | Very-Likely |
| 3 | Gate-Ox-Time-Short | → | P-Ch-Conc-Low | Very-Likely |
| 3 | Gate-Ox-Time-Very-Long | → | P-Ch-Conc-High | Likely |
| 3 | Gate-Ox-Time-Very-Short | → | P-Ch-Conc-Low | Very-Likely |
| 3 | Poly-Gate-Over-Etch | → | Channel-Length-Short | Maybe |
| 3 | Poly-Gate-Over-Etch | → | Channel-Length-Very-Short | Very-Likely |
| 3 | Poly-Gate-Over-Expose | → | Channel-Length-Short | Very-Likely |
| 3 | Poly-Gate-Over-Expose | → | Channel-Length-Very-Short | Very-Likely |
| 3 | Poly-Gate-Under-Etch | → | Channel-Length-Long | Maybe |
| 3 | Poly-Gate-Under-Etch | → | Channel-Length-Very-Long | Very-Likely |
| 3 | Poly-Gate-Under-Expose | → | Channel-Length-Long | Very-Likely |
| 3 | Poly-Gate-Under-Expose | → | Channel-Length-Very-Long | Very-Likely |

# Appendix E:  Generic Rules

*;;;; Generic Rules for the causal associations in the AESOP PIES knowledge base.*
*;;;; John L. Mohammed, 1994*

*;;; Generic Rules for Type 1 causal associations: phenomenon captured by models,*
*;;; and no topological/catastrophic changes involved.*

*;; Oxide thicknesses depend on Oxidation temperatures and times*
```
(defGenericRule Ox-Temp-Determines-Ox-Thickness
  :variables
  (($oxidation-step |:| time-interval)
   ($oxide-layer    |:| layer))
  :conditions
  ((:OR
    (is-type das-oxidation (process-at $oxidation-step))
    (is-type das-ox/drive-in (process-at $oxidation-step)))
   (created-during $oxide-layer $oxidation-step)
   (is-type oxide
            (@ (material (any-member
                          (@ (layer-regions $oxide-layer)
                             (end-of $oxidation-step))))
               (end-of $oxidation-step))))
  :rule-patterns
  (
   (((parameter $temperature $oxidation-step) HIGH) -->
    ((thickness $oxide-layer) THICK) MAYBE)
   (((parameter $temperature $oxidation-step) LOW) -->
    ((thickness $oxide-layer) THIN) MAYBE)
   (((parameter $temperature $oxidation-step) VERY-HIGH) -->
    ((thickness $oxide-layer) VERY-THICK) MAYBE)
   (((parameter $temperature $oxidation-step) VERY-LOW) -->
    ((thickness $oxide-layer) VERY-THIN) MAYBE)
   (((parameter $duration $oxidation-step) LONG) -->
    ((thickness $oxide-layer) THICK) VERY-LIKELY)
   (((parameter $duration $oxidation-step) SHORT) -->
    ((thickness $oxide-layer) THIN) VERY-LIKELY)
   (((parameter $duration $oxidation-step) VERY-LONG) -->
    ((thickness $oxide-layer) VERY-THICK) VERY-LIKELY)
   (((parameter $duration $oxidation-step) VERY-SHORT) -->
    ((thickness $oxide-layer) VERY-THIN) VERY-LIKELY)
   ))
```

```
;;  Initial concentrations and implant doses determine implant concentrations
;; Same dopant species
(defGenericRule Implant-Dose-Affects-Concentration-1
  :variables
  (($implant-step |:| time-interval)
   ($old-layer     e  (new-layers $implant-step)))
  :conditions
  (;; The step is an implantation step
   (is-type das-ion-implantation (process-at $implant-step))
   (is-type silicon
            (@ (material (any-member (@ (layer-regions $old-layer)
                                        (start-of $implant-step))))
               (start-of $implant-step)))
   ;; The implant element type is the same as the existing dominant dopant
   (V=
    (@ (dopant-type (parameter $dopant $implant-step))
       (start-of $implant-step))
    (@ (dopant-type (dopant (any-member (@ (layer-regions $old-layer)
                                           (start-of $implant-step)))))
       (start-of $implant-step))))
  :rule-patterns
  (
   ((($dose $implant-step) HIGH)      -->
    ((concentration
      (@ (layer (any-member (new-layer $old-layer $implant-step)))
         (end-of $implant-step))) HIGH) MUST)
   ((($dose $implant-step) LOW)       -->
    ((concentration
      (@ (layer (any-member (new-layer $old-layer $implant-step)))
         (end-of $implant-step))) LOW) MUST)
   ((($dose $implant-step) VERY-HIGH) -->
    ((concentration
      (@ (layer (any-member (new-layer $old-layer $implant-step)))
         (end-of $implant-step))) VERY-HIGH) MUST)
   ((($dose $implant-step) VERY-LOW)  -->
    ((concentration
      (@ (layer (any-member (new-layer $old-layer $implant-step)))
         (end-of $implant-step))) VERY-LOW) MUST)
   (((concentration $old-layer) HIGH) -->
    ((concentration
      (@ (layer (any-member (new-layer $old-layer $implant-step)))
         (end-of $implant-step))) HIGH) VERY-LIKELY)
   (((concentration $old-layer) LOW)  -->
    ((concentration
      (@ (layer (any-member (new-layer $old-layer $implant-step)))
         (end-of $implant-step))) LOW) VERY-LIKELY)
   (((concentration $old-layer) VERY-HIGH) -->
    ((concentration
      (@ (layer (any-member (new-layer $old-layer $implant-step)))
         (end-of $implant-step))) VERY-HIGH) VERY-LIKELY)
   (((concentration $old-layer) VERY-LOW)  -->
    ((concentration
      (@ (layer (any-member (new-layer $old-layer $implant-step)))
         (end-of $implant-step))) VERY-LOW) VERY-LIKELY)
   ))
```

```
;; Opposite dopant species, implant does not change dominant dopant type
(defGenericRule Implant-Dose-Affects-Concentration-2
  :variables (($implant-step |:| time-interval)
              ($old-layer     e  (new-layers $implant-step)))
  :conditions
  ((is-type das-ion-implantation (process-at $implant-step))
   (is-type silicon
            (@ (material (any-member (@ (layer-regions $old-layer)
                                        (start-of $implant-step))))
               (start-of $implant-step)))
   ;; The implant element type is not the same as the existing dominant dopant
   (:NOT (V= (@ (dopant-type (parameter $dopant $implant-step))
                (start-of $implant-step))
             (@ (dopant-type
                  (dopant (any-member (@ (layer-regions $old-layer)
                                        (start-of $implant-step)))))
                (start-of $implant-step))))
   ;; The implant doesn't change the dominant dopant type
   (V= (@ (dopant-type
            (dopant (any-member (new-layer $old-layer $implant-step))))
          (end-of $implant-step))
       (@ (dopant-type
            (dopant (any-member (@ (layer-regions $old-layer)
                                  (start-of $implant-step)))))
          (start-of $implant-step))))
  :rule-patterns
  (((($dose $implant-step) HIGH)      -->
    ((concentration
      (@ (layer (any-member (new-layer $old-layer $implant-step)))
         (end-of $implant-step))) LOW)       VERY-LIKELY)
   ((($dose $implant-step) LOW)       -->
    ((concentration
      (@ (layer (any-member (new-layer $old-layer $implant-step)))
         (end-of $implant-step))) HIGH)      VERY-LIKELY)
   ((($dose $implant-step) VERY-HIGH) -->
    ((concentration
      (@ (layer (any-member (new-layer $old-layer $implant-step)))
         (end-of $implant-step))) VERY-LOW)  VERY-LIKELY)
   ((($dose $implant-step) VERY-LOW)  -->
    ((concentration
      (@ (layer (any-member (new-layer $old-layer $implant-step)))
         (end-of $implant-step))) VERY-HIGH) VERY-LIKELY)
   (((concentration $old-layer) HIGH) -->
    ((concentration
      (@ (layer (any-member (new-layer $old-layer $implant-step)))
         (end-of $implant-step))) HIGH)      VERY-LIKELY)
   (((concentration $old-layer) LOW)  -->
    ((concentration
      (@ (layer (any-member (new-layer $old-layer $implant-step)))
         (end-of $implant-step))) LOW)       VERY-LIKELY)
   (((concentration $old-layer) VERY-HIGH) -->
    ((concentration
      (@ (layer (any-member (new-layer $old-layer $implant-step)))
         (end-of $implant-step))) VERY-HIGH) VERY-LIKELY)
   (((concentration $old-layer) VERY-LOW)  -->
    ((concentration
      (@ (layer (any-member (new-layer $old-layer $implant-step)))
         (end-of $implant-step))) VERY-LOW)  VERY-LIKELY)))
```

```
;; Opposite dopant species, implant does change dominant dopant type
(defGenericRule Implant-Dose-Affects-Concentration-3
  :variables (($implant-step |:| time-interval)
              ($old-layer     e  (new-layers $implant-step)))
  :conditions
  ((is-type das-ion-implantation (process-at $implant-step))
   (is-type silicon
            (@ (material (any-member (@ (layer-regions $old-layer)
                                        (start-of $implant-step))))
               (start-of $implant-step)))
   ;; The old-layer is not undoped
   (:NOT (V= (@ (dopant (any-member (@ (layer-regions $old-layer)
                                       (start-of $implant-step))))
                (start-of $implant-step))
             *NO-DOPANT*))
   ;; The implant element type is not the same as the existing dominant dopant
   (:NOT (V= (@ (dopant-type (parameter $dopant $implant-step))
                (start-of $implant-step))
             (@ (dopant-type
                 (dopant (any-member (@ (layer-regions $old-layer)
                                        (start-of $implant-step)))))
                (start-of $implant-step))))
   ;; The implant changes the dominant dopant type
   (:NOT
    (V= (@ (dopant-type
            (dopant (any-member (new-layer $old-layer $implant-step))))
           (end-of $implant-step))
        (@ (dopant-type
            (dopant (any-member (@ (layer-regions $old-layer)
                                   (end-of $implant-step)))))
           (end-of $implant-step)))))
  :rule-patterns
  (((($dose $implant-step) HIGH)      -->
    ((concentration
      (@ (layer (any-member (new-layer $old-layer $implant-step)))
         (end-of $implant-step))) HIGH)      VERY-LIKELY)
   ((($dose $implant-step) LOW)       -->
    ((concentration
      (@ (layer (any-member (new-layer $old-layer $implant-step)))
         (end-of $implant-step))) LOW)       VERY-LIKELY)
   ((($dose $implant-step) VERY-HIGH) -->
    ((concentration
      (@ (layer (any-member (new-layer $old-layer $implant-step)))
         (end-of $implant-step))) VERY-HIGH) VERY-LIKELY)
   ((($dose $implant-step) VERY-LOW)  -->
    ((concentration
      (@ (layer (any-member (new-layer $old-layer $implant-step)))
         (end-of $implant-step))) VERY-LOW)  VERY-LIKELY)
   (((concentration $old-layer) HIGH) -->
    ((concentration
      (@ (layer (any-member (new-layer $old-layer $implant-step)))
         (end-of $implant-step))) LOW)       VERY-LIKELY)
   (((concentration $old-layer) LOW)  -->
    ((concentration
      (@ (layer (any-member (new-layer $old-layer $implant-step)))
         (end-of $implant-step))) HIGH)      VERY-LIKELY)
   (((concentration $old-layer) VERY-HIGH) -->
    ((concentration
```

```
      (@ (layer (any-member (new-layer $old-layer $implant-step)))
         (end-of $implant-step))) VERY-LOW)  VERY-LIKELY)
   (((concentration $old-layer) VERY-LOW)  -->
    ((concentration
      (@ (layer (any-member (new-layer $old-layer $implant-step)))
         (end-of $implant-step))) VERY-HIGH) VERY-LIKELY)))
```

```
(defGenericRule Initial-Doping-Affects-Final
  :variables (($layer |:| layer))
  :conditions
  (;; The layer existed at the beginning of the process
   (exists-at $layer start)
   ;; The layer was initially doped
   (:NOT (v= (@ (dopant
                 (any-member (@ (layer-regions $layer) start))) start)
             *NO-DOPANT*))
   ;; The layer survives to the end of the process
   (exists-at $layer end))
  :rule-patterns
  ((((start-concentration $layer) HIGH) -->
    ((concentration $layer) HIGH) VERY-LIKELY)
   (((start-concentration $layer) LOW) -->
    ((concentration $layer) LOW) VERY-LIKELY)
   (((start-concentration $layer) VERY-HIGH) -->
    ((concentration $layer) VERY-HIGH) VERY-LIKELY)
   (((start-concentration $layer) VERY-LOW) -->
    ((concentration $layer) VERY-LOW) VERY-LIKELY)))
```

```
;;; Generic Rules for Type 2 causal associations: phenomenon captured by models,
;;; but topological/catastrophic changes involved.

;; If the substrate dopant type is initially wrong, it is wrong forever.
(defGenericRule Initial-Dopant-Type-Affects-Final
  :variables (($layer |:| layer))
  :conditions
  (;; The layer existed at the beginning of the process
   (exists-at $layer start)
   ;; The layer was initially doped
   (:NOT (v= (@ (dopant
                 (any-member (@ (layer-regions $layer) start))) start)
             *NO-DOPANT*))
   ;; The layer survives to the end of the process
   (exists-at $layer end))
  :rule-patterns
  ((((start-dopant-type $layer) wrong-type) -->
    ((dopant-type $layer) wrong-type) VERY-LIKELY)))
```

```
;; An oxide that normally survives to the end of the process will not be present
;; if the oxidation step that creates it is not performed.
(defGenericRule Oxidation-Step-Omitted
  :variables (($oxidation-step |:| time-interval)
              ($oxide-layer |:| layer))
  :conditions
  (;; The step was an oxidation step
   (:OR (is-type das-oxidation (process-at $oxidation-step))
        (is-type das-ox/drive-in (process-at $oxidation-step)))
   ;; The oxide layer was created during it.
   (created-during $oxide-layer $oxidation-step)
   (is-type oxide
            (@ (material (any-member (@ (layer-regions $oxide-layer)
                                        (end-of $oxidation-step))))
               (end-of $oxidation-step)))
   ;; The layer normally survives to the end of the process
   (exists-at $oxide-layer end))
  :rule-patterns
  ((($oxidation-step not-done) -->
    ($oxide-layer non-existent) VERY-LIKELY)))



;; If an implant creates a structure by changing the dominant dopant type,
;; then the structure will not exist if the implant is not performed, or if
;; the wrong species is implanted.
(defGenericRule Implant-Step-Omitted
  :variables (($implant-step |:| time-interval)
              ($old-layer     e  (new-layers $implant-step)))
  :conditions
  ((is-type das-ion-implantation (process-at $implant-step))
   ;; The old-layer is doped
   (:NOT (V= (@ (dopant-type
                 (dopant (any-member (@ (layer-regions $old-layer)
                                        (start-of $implant-step)))))
                (start-of $implant-step))
             *NO-DOPANT*))
   ;; The implant element type is not the same as the existing dominant dopant
   (:NOT (V= (@ (dopant-type (parameter $dopant $implant-step))
                (start-of $implant-step))
             (@ (dopant-type
                 (dopant (any-member (@ (layer-regions $old-layer)
                                        (start-of $implant-step)))))
                (start-of $implant-step))))
   ;; The implant changes the dominant dopant type
   (V= (@ (dopant-type (parameter $dopant $implant-step))
          (start-of $implant-step))
       (@ (dopant-type
           (dopant (any-member (new-layer $old-layer $implant-step))))
          (end-of $implant-step))))
  :rule-patterns
  ((($implant-step not-done) -->
    ((@ (layer (any-member (new-layer $old-layer $implant-step)))
        (end-of $implant-step)) non-existent) VERY-LIKELY)
   (((parameter $dopant $implant-step) wrong-type) -->
    ((@ (layer (any-member (new-layer $old-layer $implant-step)))
        (end-of $implant-step)) non-existent) MAYBE)))
```

*;; If a normally masked implant that creates a structure by changing the dominant*
*;; dopant type is not masked, then the dopant type will be wrong in the regions*
*;; that would normally be masked.*

```
(defGenericRule Implant-Not-Masked-1
  :variables (($implant-step |:| time-interval)
              ($old-layer     e  (new-layers $implant-step)))
  :conditions
  ((is-type das-ion-implantation (process-at $implant-step))
   ;; The implant element type is not the same as the existing dominant dopant
   (:NOT (V= (@ (dopant-type (parameter $dopant $implant-step))
                (start-of $implant-step))
             (@ (dopant-type
                 (dopant (any-member (@ (layer-regions $old-layer)
                                        (start-of $implant-step)))))
                (start-of $implant-step))))
   ;; The implant changes the dominant dopant type
   (V= (@ (dopant-type (parameter $dopant $implant-step))
          (start-of $implant-step))
       (@ (dopant-type
           (dopant (any-member (new-layer $old-layer $implant-step))))
          (end-of $implant-step)))
   ;; The implant is normally masked
   (exists $lr e (@ (layer-regions $old-layer) (start-of $implant-step))
     (is-type photoresist (@ (material (surface (region-type $lr)))
                             (start-of $implant-step)))))
  :rule-patterns
  ((($implant-step not-masked) --> ((dopant-type $old-layer) wrong-type)
    Maybe)))
```

*;;; Generic Rules for Type 3 causal associations: phenomenon not captured by models,*
*;;; and may or may not involve topological/catastrophic changes.*

*;; Intra-level rule for rules concerning effect of lateral changes*
*;; in dimensions of self-aligned structures.*

```
(defGenericRule Structure-Masked-Implant
  :variables
  (($implant-step       |:| time-interval)
   ($masking-layer       e  (new-layers $implant-step))
   ($masked-layer        e  (new-layers $implant-step))
   ($masking-structure   e  (@ (layer-regions $masking-layer)
                               (start-of $implant-step)))
   ($masked-structure    e  (@ (layer-regions $masked-layer)
                               (start-of $implant-step)))
   ($implanted-region    e  (@ (regions (region-type $masking-structure))
                               (start-of $implant-step))))
  :conditions
  ((is-type das-ion-implantation (process-at $implant-step))
   (:NOT (V= $masking-layer $masked-layer))
   (V= (@ (region-type $masking-structure) (start-of $implant-step))
       (@ (region-type $masked-structure)  (start-of $implant-step)))
   (:NOT (V= (@ (left $implanted-region) (start-of $implant-step))
             REGION-EDGE))
   (:NOT (V= (@ (right $implanted-region) (start-of $implant-step))
             REGION-EDGE))
   (:NOT (is-type photoresist
          (@ (material (surface (region-type $implanted-region)))
             (start-of $implant-step))))
   (:NOT (is-type photoresist
          (@ (material (surface (region-type (left $implanted-region))))
             (start-of $implant-step))))
   (:NOT (is-type photoresist
          (@ (material
              (surface (region-type (right $implanted-region))))
             (start-of $implant-step))))
   (< (ion-bottom (@ (region-type $implanted-region)
                     (start-of $implant-step))
                  $implant-step)
      (@ (top-pos $masking-structure) (start-of $implant-step)))
   (> (ion-bottom (@ (region-type $implanted-region)
                     (start-of $implant-step))
                  $implant-step)
      (@ (top-pos $masked-structure) (start-of $implant-step)))
   (exists $lr e (new-layer $masked-layer $implant-step)
    (V= (@ (region-type $lr) (end-of $implant-step))
        (@ (region-type (left $implanted-region))
           (end-of $implant-step))))
   (exists $lr e (new-layer $masked-layer $implant-step)
    (V= (@ (region-type $lr) (end-of $implant-step))
        (@ (region-type (right $implanted-region))
           (end-of $implant-step)))))
   :rule-patterns
   ((((width $masking-layer in $implanted-region) HIGH) -->
     ((width $masked-layer in $implanted-region) HIGH) VERY-LIKELY)
    (((width $masking-layer in $implanted-region) LOW)  -->
     ((width $masked-layer in $implanted-region) LOW)  VERY-LIKELY)))
```

```
(defGenericRule Insufficient/Excess-Lateral-Etch
  :variables
  (($etch-step    |:| time-interval)
   ($layer         e  (changed-layers $etch-step))
   ($layer-region  e  (@ (layer-regions $layer) (start-of $etch-step)))
   ($wr            e  (@ (regions (region-type $layer-region)) (start-of
$etch-step))))
  :conditions
  (
   ;; Condition 1: The step is an etch step involving lateral etching
   (is-type das-etch (process-at $etch-step))
   (< (parameter $etch-angle $etch-step) etch-angle-threshold)
   (:NOT (V= (@ (left $wr) (start-of $etch-step)) REGION-EDGE))
   (:NOT (V= (@ (right $wr) (start-of $etch-step)) REGION-EDGE))

   ;; Condition 2: The layer-region survives the etch step
   (:NOT (destroyed $layer-region (end-of $etch-step)))

   ;; Condition 3: Those around it don't, and the step involves overetching
   (exists $lr e (@ (layer-regions $layer) (start-of $etch-step))
    (:AND
     (V= (@ (region-type $lr) (start-of $etch-step))
         (@ (region-type (left $wr)) (start-of $etch-step)))
     (> (parameter $duration $etch-step)
        (etch-destroy-time $lr $etch-step))))

   (exists $lr e (@ (layer-regions $layer) (start-of $etch-step))
    (:AND
     (V= (@ (region-type $lr) (start-of $etch-step))
         (@ (region-type (right $wr)) (start-of $etch-step)))
     (> (parameter $duration $etch-step)
        (etch-destroy-time $lr $etch-step))))
   )
  :rule-patterns
  (
   (($etch-step under-etch) --> ((width $layer in $wr) HIGH) MAYBE)
   (($etch-step under-etch) --> ((width $layer in $wr) VERY-HIGH)
    VERY-LIKELY)
   (($etch-step over-etch)  --> ((width $layer in $wr) LOW) MAYBE)
   (($etch-step over-etch)  --> ((width $layer in $wr) VERY-LOW)
    VERY-LIKELY)
   ))



;; Insufficient/excess exposure of resist can change the lateral dimensions of
;; the structures patterned by the resist.
(defGenericRule Insufficient/Excess-Photoresist-Exposure
  :variables
  (($expose-step  |:| time-interval)
   ($region-type  |:| region-type)
   ($wr            e  (@ (regions $region-type) (end-of $expose-step))))
  :conditions
  (
   (is-type das-mask-expose (process-at $expose-step))

   (is-type photoresist (@ (material (surface $region-type))
                           (end-of $expose-step)))
```

```
  (:NOT (is-soft-photoresist? (@ (material (surface $region-type))
                                 (end-of $expose-step))))

  (:NOT (V= (@ (left $wr) (start-of $expose-step)) REGION-EDGE))
  (:NOT (V= (@ (right $wr) (start-of $expose-step)) REGION-EDGE))

  (is-soft-photoresist?
   (@ (material (surface (region-type (left $wr))))
      (end-of $expose-step)))
  (is-soft-photoresist?
   (@ (material (surface (region-type (right $wr))))
      (end-of $expose-step)))
  )
 :rule-patterns
 (
  (($expose-step under-expose) --> ((width photoresist in $wr) HIGH)
   MAYBE)
  (($expose-step under-expose) -->
   ((width photoresist in $wr) VERY-HIGH) VERY-LIKELY)
  (($expose-step over-expose)  --> ((width photoresist in $wr) LOW)
   MAYBE)
  (($expose-step over-expose)  -->
   ((width photoresist in $wr) VERY-LOW) VERY-LIKELY)
  ))
```

*;; Intra-level rule for transfer of incorrect lateral dimension in resist to*
*;; layer patterned by resist.*

```
(defGenericRule Resist-Masked-Etch
  :variables
  (($etch-step    |:| time-interval)
   ($etched-layer  e  (changed-layers $etch-step))
   ($etched-lr     e  (@ (layer-regions $etched-layer)
                         (start-of $etch-step)))
   ($wr            e  (@ (regions (region-type $etched-lr))
                         (start-of $etch-step))))
  :conditions
  (
   (is-type das-etch (process-at $etch-step))

   (exists-at $etched-lr (start-of $etch-step))
   (:NOT (is-type photoresist (@ (material $etched-lr)
                                 (start-of $etch-step))))
   (is-type photoresist (@ (material (surface (region-type $etched-lr)))
                           (start-of $etch-step)))

   (:NOT (V= (@ (left $wr) (start-of $etch-step)) REGION-EDGE))
   (:NOT (V= (@ (right $wr) (start-of $etch-step)) REGION-EDGE))

   (exists $lr e (@ (layer-regions $etched-layer) (start-of $etch-step))
    (:AND
     (V= (@ (region-type $lr) (start-of $etch-step))
         (@ (region-type (left $wr)) (start-of $etch-step)))
     (destroyed $lr (end-of $etch-step))))
   (exists $lr e (@ (layer-regions $etched-layer) (start-of $etch-step))
    (:AND
     (V= (@ (region-type $lr) (start-of $etch-step))
```

```
         (@ (region-type (right $wr)) (start-of $etch-step)))
     (destroyed $lr (end-of $etch-step))))
   )
  :rule-patterns
  (
   (((width photoresist in $wr) HIGH) -->
    ((width $etched-layer in $wr) HIGH) MUST)
   (((width photoresist in $wr) LOW)  -->
    ((width $etched-layer in $wr) LOW)  MUST)
   (((width photoresist in $wr) VERY-HIGH) -->
    ((width $etched-layer in $wr) VERY-HIGH) MUST)
   (((width photoresist in $wr) VERY-LOW)  -->
    ((width $etched-layer in $wr) VERY-LOW)  MUST)
   ))
```

*;; When a silicon layer containing Boron or Arsenic is partially oxidized, some of the boron*
*;; is lost into the oxide, while the arsenic "piles up" in the silicon at the interface.*
*;; If the dominant dopant type is N, then the concentration increases. else it decreases.*

```
(defGenericRule Thermal-Oxidation-Redistributes-Impurities-N
  :variables
  (($oxidation-step |:| time-interval)
   ($doped-layer    |:| layer)
   ($doped-lr        e  (@ (layer-regions $doped-layer)
                           (end-of $oxidation-step))))
  :conditions
  ((:OR (is-type das-oxidation (process-at $oxidation-step))
        (is-type das-ox/drive-in (process-at $oxidation-step)))
   (exists-at $doped-lr (end-of $oxidation-step))
   (V= |Silicon| (@ (material $doped-lr) (start-of $oxidation-step)))
   (uppermost-oxidation-survivor $doped-lr $oxidation-step)
   (is-type N-dopant (@ (dopant $doped-lr) (start-of $oxidation-step)))
   (> (parameter $duration $oxidation-step)
      (oxidation-destroy-time
       (@ (above $doped-lr) (start-of $oxidation-step))
       (parameter $temperature $oxidation-step)
       (start-of $oxidation-step))))
  :rule-patterns
  (
   (((parameter $duration $oxidation-step) LONG) -->
    ((concentration $doped-layer) HIGH) VERY-LIKELY)
   (((parameter $duration $oxidation-step) VERY-LONG) -->
    ((concentration $doped-layer) HIGH) LIKELY)
   (((parameter $duration $oxidation-step) SHORT) -->
    ((concentration $doped-layer) LOW) VERY-LIKELY)
   (((parameter $duration $oxidation-step) VERY-SHORT) -->
    ((concentration $doped-layer) LOW) VERY-LIKELY)
   ))
```

```
(defGenericRule Thermal-Oxidation-Redistributes-Impurities-P
  :variables
  (($oxidation-step |:| time-interval)
   ($doped-layer    |:| layer)
   ($doped-lr        e  (@ (layer-regions $doped-layer)
                           (end-of $oxidation-step))))
  :conditions
  ((:OR (is-type das-oxidation (process-at $oxidation-step))
        (is-type das-ox/drive-in (process-at $oxidation-step)))
   (exists-at $doped-lr (end-of $oxidation-step))
   (V= |Silicon| (@ (material $doped-lr) (start-of $oxidation-step)))
   (uppermost-oxidation-survivor $doped-lr $oxidation-step)
   (is-type P-dopant (@ (dopant $doped-lr) (start-of $oxidation-step)))
   (> (parameter $duration $oxidation-step)
      (oxidation-destroy-time
       (@ (above $doped-lr) (start-of $oxidation-step))
       (parameter $temperature $oxidation-step)
       (start-of $oxidation-step))))
  :rule-patterns
  (
   (((parameter $duration $oxidation-step) LONG) -->
    ((concentration $doped-layer) LOW) VERY-LIKELY)
   (((parameter $duration $oxidation-step) VERY-LONG) -->
    ((concentration $doped-layer) LOW) LIKELY)
   (((parameter $duration $oxidation-step) SHORT) -->
    ((concentration $doped-layer) HIGH) VERY-LIKELY)
   (((parameter $duration $oxidation-step) VERY-SHORT) -->
    ((concentration $doped-layer) HIGH) VERY-LIKELY)
   ))
```

# Appendix F: Associations from Generic Rules

This appendix lists the instantiations of the Generic Rules listed in Appendix E that were obtained by applying the rules to a very simplified representation of the CMOS process.

## Legend

### Processes:

Oxidation Processes:

```
Oxidation[1]          Gate Oxidation
```

Implantation Processes:

```
Implant[5]            N+ Source/Drain Implant
Implant[4]            P+ Source/Drain Implant
Implant[3]            N-Channel Implant
Implant[2]            P-Channel Implant
Implant[1]            N-Well Implant
```

Etch Processes:

```
Etch[1]               Poly Gate Etch
```

Photolithography Processes:

```
"Select Mask"[5]      N+Source/Drain mask
"Select Mask"[4]      P+Source/Drain mask
"Select Mask"[3]      Gate mask
"Select Mask"[2]      N-Channel mask
"Select Mask"[1]      N-Well mask
```

### Wafer Regions appearing in rules:

```
Wafer-Region-4 = PMOS gate region
Wafer-Region-7 = NMOS gate region
```

## Numbered Layers:

```
No.  Name                    Material      Dopant
 1 = Layer-1                 Resist-1813
 2 = N-Well-Layer            Silicon       Arsenic
 3 = Layer-3                 Silicon       Boron
 4 = P-Channel-Layer         Silicon       Arsenic
 5 = Layer-5                 Resist-1813
 6 = N-Channel-Layer         Silicon       Boron
 7 = Gate-Oxide-Layer        Oxide
 8 = Gate-Oxide-Layer        Oxide
 9 = Polysilicon-Layer       Polysilicon
10 = Layer-10                Resist-1813
11 = Layer-11                Resist-1813
12 = P+Doped-Poly-Layer      Polysilicon   Boron
13 = P+Source/Drain-Layer    Silicon       Boron
14 = Layer-14                Resist-1813
15 = N+Doped-Poly-Layer      Polysilicon   Arsenic
16 = N+Source/Drain-Layer    Silicon       Arsenic
```

## Forms appearing in rules:

```
(@ (LAYER (ANY-MEMBER (NEW-LAYER SUBSTRATE |Implant[2]|)))
   (END-OF |Implant[2]|)) = LAYER-3
(@ (LAYER (ANY-MEMBER (NEW-LAYER LAYER-3 |Implant[3]|)))
   (END-OF |Implant[3]|)) = N-CHANNEL-LAYER
(@ (LAYER (ANY-MEMBER (NEW-LAYER N-WELL-LAYER |Implant[2]|)))
   (END-OF |Implant[2]|)) = P-CHANNEL-LAYER
(@ (LAYER (ANY-MEMBER (NEW-LAYER SUBSTRATE |Implant[1]|)))
   (END-OF |Implant[1]|)) = N-WELL-LAYER
(@ (LAYER (ANY-MEMBER
           (NEW-LAYER P-CHANNEL-LAYER |Implant[4]|)))
   (END-OF |Implant[4]|)) = P+SOURCE/DRAIN-LAYER
(@ (LAYER (ANY-MEMBER
           (NEW-LAYER POLYSILICON-LAYER |Implant[4]|)))
   (END-OF |Implant[4]|)) = P+DOPED-POLY-LAYER
(@ (LAYER (ANY-MEMBER
           (NEW-LAYER N-CHANNEL-LAYER |Implant[5]|)))
   (END-OF |Implant[5]|)) = N+SOURCE/DRAIN-LAYER
(@ (LAYER (ANY-MEMBER
           (NEW-LAYER POLYSILICON-LAYER |Implant[5]|)))
   (END-OF |Implant[5]|)) = N+DOPED-POLY-LAYER
```

## Category 1 Causal Associations

```
Running rule OX-TEMP-DETERMINES-OX-THICKNESS -- Executed in 0.4 seconds
Running rule OX-TEMP-DETERMINES-OX-THICKNESS produces 16 rules:

((((PARAMETER $DURATION |Oxidation[1]|) VERY-SHORT) -->
  ((THICKNESS LAYER-7) VERY-THIN) VERY-LIKELY)
 (((PARAMETER $DURATION |Oxidation[1]|) VERY-LONG) -->
  ((THICKNESS LAYER-7) VERY-THICK) VERY-LIKELY)
 (((PARAMETER $DURATION |Oxidation[1]|) SHORT) -->
  ((THICKNESS LAYER-7) THIN) VERY-LIKELY)
 (((PARAMETER $DURATION |Oxidation[1]|) LONG) -->
  ((THICKNESS LAYER-7) THICK) VERY-LIKELY)
 (((PARAMETER $TEMPERATURE |Oxidation[1]|) VERY-LOW) -->
  ((THICKNESS LAYER-7) VERY-THIN) MAYBE)
 (((PARAMETER $TEMPERATURE |Oxidation[1]|) VERY-HIGH) -->
  ((THICKNESS LAYER-7) VERY-THICK) MAYBE)
 (((PARAMETER $TEMPERATURE |Oxidation[1]|) LOW) -->
  ((THICKNESS LAYER-7) THIN) MAYBE)
 (((PARAMETER $TEMPERATURE |Oxidation[1]|) HIGH) -->
  ((THICKNESS LAYER-7) THICK) MAYBE)
 (((PARAMETER $DURATION |Oxidation[1]|) VERY-SHORT) -->
  ((THICKNESS LAYER-8) VERY-THIN) VERY-LIKELY)
 (((PARAMETER $DURATION |Oxidation[1]|) VERY-LONG) -->
  ((THICKNESS LAYER-8) VERY-THICK) VERY-LIKELY)
 (((PARAMETER $DURATION |Oxidation[1]|) SHORT) -->
  ((THICKNESS LAYER-8) THIN) VERY-LIKELY)
 (((PARAMETER $DURATION |Oxidation[1]|) LONG) -->
  ((THICKNESS LAYER-8) THICK) VERY-LIKELY)
 (((PARAMETER $TEMPERATURE |Oxidation[1]|) VERY-LOW) -->
  ((THICKNESS LAYER-8) VERY-THIN) MAYBE)
 (((PARAMETER $TEMPERATURE |Oxidation[1]|) VERY-HIGH) -->
  ((THICKNESS LAYER-8) VERY-THICK) MAYBE)
 (((PARAMETER $TEMPERATURE |Oxidation[1]|) LOW) -->
  ((THICKNESS LAYER-8) THIN) MAYBE)
 (((PARAMETER $TEMPERATURE |Oxidation[1]|) HIGH) -->
  ((THICKNESS LAYER-8) THICK) MAYBE))
```

```
Running rule IMPLANT-DOSE-AFFECTS-CONCENTRATION-1 -- Executed in 0.4
seconds
Running rule IMPLANT-DOSE-AFFECTS-CONCENTRATION-1 produces 16 rules:

((((CONCENTRATION SUBSTRATE) VERY-LOW) -->
  ((CONCENTRATION
    (@ (LAYER (ANY-MEMBER (NEW-LAYER SUBSTRATE |Implant[2]|)))
       (END-OF |Implant[2]|)))
   VERY-LOW)
  VERY-LIKELY)
 (((CONCENTRATION SUBSTRATE) VERY-HIGH) -->
  ((CONCENTRATION
    (@ (LAYER (ANY-MEMBER (NEW-LAYER SUBSTRATE |Implant[2]|)))
       (END-OF |Implant[2]|)))
   VERY-HIGH)
  VERY-LIKELY)
 (((CONCENTRATION SUBSTRATE) LOW) -->
  ((CONCENTRATION
    (@ (LAYER (ANY-MEMBER (NEW-LAYER SUBSTRATE |Implant[2]|)))
       (END-OF |Implant[2]|)))
   LOW)
  VERY-LIKELY)
 (((CONCENTRATION SUBSTRATE) HIGH) -->
  ((CONCENTRATION
    (@ (LAYER (ANY-MEMBER (NEW-LAYER SUBSTRATE |Implant[2]|)))
       (END-OF |Implant[2]|)))
   HIGH)
  VERY-LIKELY)
 ((($DOSE |Implant[2]|) VERY-LOW) -->
  ((CONCENTRATION
    (@ (LAYER (ANY-MEMBER (NEW-LAYER SUBSTRATE |Implant[2]|)))
       (END-OF |Implant[2]|)))
   VERY-LOW)
  MUST)
 ((($DOSE |Implant[2]|) VERY-HIGH) -->
  ((CONCENTRATION
    (@ (LAYER (ANY-MEMBER (NEW-LAYER SUBSTRATE |Implant[2]|)))
       (END-OF |Implant[2]|)))
   VERY-HIGH)
  MUST)
 ((($DOSE |Implant[2]|) LOW) -->
  ((CONCENTRATION
    (@ (LAYER (ANY-MEMBER (NEW-LAYER SUBSTRATE |Implant[2]|)))
       (END-OF |Implant[2]|)))
   LOW)
  MUST)
 ((($DOSE |Implant[2]|) HIGH) -->
  ((CONCENTRATION
    (@ (LAYER (ANY-MEMBER (NEW-LAYER SUBSTRATE |Implant[2]|)))
       (END-OF |Implant[2]|)))
   HIGH)
  MUST)
 (((CONCENTRATION LAYER-3) VERY-LOW) -->
  ((CONCENTRATION
    (@ (LAYER (ANY-MEMBER (NEW-LAYER LAYER-3 |Implant[3]|)))
       (END-OF |Implant[3]|)))
   VERY-LOW)
  VERY-LIKELY)
```

```
(((CONCENTRATION LAYER-3) VERY-HIGH) -->
 ((CONCENTRATION
   (@ (LAYER (ANY-MEMBER (NEW-LAYER LAYER-3 |Implant[3]|)))
      (END-OF |Implant[3]|)))
  VERY-HIGH)
 VERY-LIKELY)
(((CONCENTRATION LAYER-3) LOW) -->
 ((CONCENTRATION
   (@ (LAYER (ANY-MEMBER (NEW-LAYER LAYER-3 |Implant[3]|)))
      (END-OF |Implant[3]|)))
  LOW)
 VERY-LIKELY)
(((CONCENTRATION LAYER-3) HIGH) -->
 ((CONCENTRATION
   (@ (LAYER (ANY-MEMBER (NEW-LAYER LAYER-3 |Implant[3]|)))
      (END-OF |Implant[3]|)))
  HIGH)
 VERY-LIKELY)
((($DOSE |Implant[3]|) VERY-LOW) -->
 ((CONCENTRATION
   (@ (LAYER (ANY-MEMBER (NEW-LAYER LAYER-3 |Implant[3]|)))
      (END-OF |Implant[3]|)))
  VERY-LOW)
 MUST)
((($DOSE |Implant[3]|) VERY-HIGH) -->
 ((CONCENTRATION
   (@ (LAYER (ANY-MEMBER (NEW-LAYER LAYER-3 |Implant[3]|)))
      (END-OF |Implant[3]|)))
  VERY-HIGH)
 MUST)
((($DOSE |Implant[3]|) LOW) -->
 ((CONCENTRATION
   (@ (LAYER (ANY-MEMBER (NEW-LAYER LAYER-3 |Implant[3]|)))
      (END-OF |Implant[3]|)))
  LOW)
 MUST)
((($DOSE |Implant[3]|) HIGH) -->
 ((CONCENTRATION
   (@ (LAYER (ANY-MEMBER (NEW-LAYER LAYER-3 |Implant[3]|)))
      (END-OF |Implant[3]|)))
  HIGH)
 MUST))
```

```
Running rule IMPLANT-DOSE-AFFECTS-CONCENTRATION-2 -- Executed in 0.2
seconds
Running rule IMPLANT-DOSE-AFFECTS-CONCENTRATION-2 produces 8 rules:

((((\$DOSE |Implant[2]|) VERY-LOW) -->
  ((CONCENTRATION
    (@ (LAYER (ANY-MEMBER (NEW-LAYER N-WELL-LAYER |Implant[2]|)))
       (END-OF |Implant[2]|)))
   VERY-HIGH)
  VERY-LIKELY)
 ((($DOSE |Implant[2]|) VERY-HIGH) -->
  ((CONCENTRATION
    (@ (LAYER (ANY-MEMBER (NEW-LAYER N-WELL-LAYER |Implant[2]|)))
       (END-OF |Implant[2]|)))
   VERY-LOW)
  VERY-LIKELY)
 ((($DOSE |Implant[2]|) LOW) -->
  ((CONCENTRATION
    (@ (LAYER (ANY-MEMBER (NEW-LAYER N-WELL-LAYER |Implant[2]|)))
       (END-OF |Implant[2]|)))
   HIGH)
  VERY-LIKELY)
 ((($DOSE |Implant[2]|) HIGH) -->
  ((CONCENTRATION
    (@ (LAYER (ANY-MEMBER (NEW-LAYER N-WELL-LAYER |Implant[2]|)))
       (END-OF |Implant[2]|)))
   LOW)
  VERY-LIKELY)
 (((CONCENTRATION N-WELL-LAYER) VERY-LOW) -->
  ((CONCENTRATION
    (@ (LAYER (ANY-MEMBER (NEW-LAYER N-WELL-LAYER |Implant[2]|)))
       (END-OF |Implant[2]|)))
   VERY-LOW)
  VERY-LIKELY)
 (((CONCENTRATION N-WELL-LAYER) VERY-HIGH) -->
  ((CONCENTRATION
    (@ (LAYER (ANY-MEMBER (NEW-LAYER N-WELL-LAYER |Implant[2]|)))
       (END-OF |Implant[2]|)))
   VERY-HIGH)
  VERY-LIKELY)
 (((CONCENTRATION N-WELL-LAYER) LOW) -->
  ((CONCENTRATION
    (@ (LAYER (ANY-MEMBER (NEW-LAYER N-WELL-LAYER |Implant[2]|)))
       (END-OF |Implant[2]|)))
   LOW)
  VERY-LIKELY)
 (((CONCENTRATION N-WELL-LAYER) HIGH) -->
  ((CONCENTRATION
    (@ (LAYER (ANY-MEMBER (NEW-LAYER N-WELL-LAYER |Implant[2]|)))
       (END-OF |Implant[2]|)))
   HIGH)
  VERY-LIKELY))
```

```
Running rule IMPLANT-DOSE-AFFECTS-CONCENTRATION-3 -- Executed in 0.3
seconds
Running rule IMPLANT-DOSE-AFFECTS-CONCENTRATION-3 produces 24 rules:

((((CONCENTRATION SUBSTRATE) VERY-LOW) -->
  ((CONCENTRATION
    (@ (LAYER (ANY-MEMBER (NEW-LAYER SUBSTRATE |Implant[1]|)))
       (END-OF |Implant[1]|)))
   VERY-HIGH)
  VERY-LIKELY)
 (((CONCENTRATION SUBSTRATE) VERY-HIGH) -->
  ((CONCENTRATION
    (@ (LAYER (ANY-MEMBER (NEW-LAYER SUBSTRATE |Implant[1]|)))
       (END-OF |Implant[1]|)))
   VERY-LOW)
  VERY-LIKELY)
 (((CONCENTRATION SUBSTRATE) LOW) -->
  ((CONCENTRATION
    (@ (LAYER (ANY-MEMBER (NEW-LAYER SUBSTRATE |Implant[1]|)))
       (END-OF |Implant[1]|)))
   HIGH)
  VERY-LIKELY)
 (((CONCENTRATION SUBSTRATE) HIGH) -->
  ((CONCENTRATION
    (@ (LAYER (ANY-MEMBER (NEW-LAYER SUBSTRATE |Implant[1]|)))
       (END-OF |Implant[1]|)))
   LOW)
  VERY-LIKELY)
 ((($DOSE |Implant[1]|) VERY-LOW) -->
  ((CONCENTRATION
    (@ (LAYER (ANY-MEMBER (NEW-LAYER SUBSTRATE |Implant[1]|)))
       (END-OF |Implant[1]|)))
   VERY-LOW)
  VERY-LIKELY)
 ((($DOSE |Implant[1]|) VERY-HIGH) -->
  ((CONCENTRATION
    (@ (LAYER (ANY-MEMBER (NEW-LAYER SUBSTRATE |Implant[1]|)))
       (END-OF |Implant[1]|)))
   VERY-HIGH)
  VERY-LIKELY)
 ((($DOSE |Implant[1]|) LOW) -->
  ((CONCENTRATION
    (@ (LAYER (ANY-MEMBER (NEW-LAYER SUBSTRATE |Implant[1]|)))
       (END-OF |Implant[1]|)))
   LOW)
  VERY-LIKELY)
 ((($DOSE |Implant[1]|) HIGH) -->
  ((CONCENTRATION
    (@ (LAYER (ANY-MEMBER (NEW-LAYER SUBSTRATE |Implant[1]|)))
       (END-OF |Implant[1]|)))
   HIGH)
  VERY-LIKELY)
 (((CONCENTRATION P-CHANNEL-LAYER) VERY-LOW) -->
  ((CONCENTRATION
    (@ (LAYER (ANY-MEMBER (NEW-LAYER P-CHANNEL-LAYER |Implant[4]|)))
       (END-OF |Implant[4]|)))
   VERY-HIGH)
  VERY-LIKELY)
```

```
(((CONCENTRATION P-CHANNEL-LAYER) VERY-HIGH) -->
 ((CONCENTRATION
   (@ (LAYER (ANY-MEMBER (NEW-LAYER P-CHANNEL-LAYER |Implant[4]|)))
      (END-OF |Implant[4]|)))
  VERY-LOW)
 VERY-LIKELY)
(((CONCENTRATION P-CHANNEL-LAYER) LOW) -->
 ((CONCENTRATION
   (@ (LAYER (ANY-MEMBER (NEW-LAYER P-CHANNEL-LAYER |Implant[4]|)))
      (END-OF |Implant[4]|)))
  HIGH)
 VERY-LIKELY)
(((CONCENTRATION P-CHANNEL-LAYER) HIGH) -->
 ((CONCENTRATION
   (@ (LAYER (ANY-MEMBER (NEW-LAYER P-CHANNEL-LAYER |Implant[4]|)))
      (END-OF |Implant[4]|)))
  LOW)
 VERY-LIKELY)
((($DOSE |Implant[4]|) VERY-LOW) -->
 ((CONCENTRATION
   (@ (LAYER (ANY-MEMBER (NEW-LAYER P-CHANNEL-LAYER |Implant[4]|)))
      (END-OF |Implant[4]|)))
  VERY-LOW)
 VERY-LIKELY)
((($DOSE |Implant[4]|) VERY-HIGH) -->
 ((CONCENTRATION
   (@ (LAYER (ANY-MEMBER (NEW-LAYER P-CHANNEL-LAYER |Implant[4]|)))
      (END-OF |Implant[4]|)))
  VERY-HIGH)
 VERY-LIKELY)
((($DOSE |Implant[4]|) LOW) -->
 ((CONCENTRATION
   (@ (LAYER (ANY-MEMBER (NEW-LAYER P-CHANNEL-LAYER |Implant[4]|)))
      (END-OF |Implant[4]|)))
  LOW)
 VERY-LIKELY)
((($DOSE |Implant[4]|) HIGH) -->
 ((CONCENTRATION
   (@ (LAYER (ANY-MEMBER (NEW-LAYER P-CHANNEL-LAYER |Implant[4]|)))
      (END-OF |Implant[4]|)))
  HIGH)
 VERY-LIKELY)
(((CONCENTRATION N-CHANNEL-LAYER) VERY-LOW) -->
 ((CONCENTRATION
   (@ (LAYER (ANY-MEMBER (NEW-LAYER N-CHANNEL-LAYER |Implant[5]|)))
      (END-OF |Implant[5]|)))
  VERY-HIGH)
 VERY-LIKELY)
(((CONCENTRATION N-CHANNEL-LAYER) VERY-HIGH) -->
 ((CONCENTRATION
   (@ (LAYER (ANY-MEMBER (NEW-LAYER N-CHANNEL-LAYER |Implant[5]|)))
      (END-OF |Implant[5]|)))
  VERY-LOW)
 VERY-LIKELY)
(((CONCENTRATION N-CHANNEL-LAYER) LOW) -->
 ((CONCENTRATION
   (@ (LAYER (ANY-MEMBER (NEW-LAYER N-CHANNEL-LAYER |Implant[5]|)))
      (END-OF |Implant[5]|)))
```

```
    HIGH)
   VERY-LIKELY)
  (((CONCENTRATION N-CHANNEL-LAYER) HIGH) -->
   ((CONCENTRATION
     (@ (LAYER (ANY-MEMBER (NEW-LAYER N-CHANNEL-LAYER |Implant[5]|)))
        (END-OF |Implant[5]|)))
    LOW)
   VERY-LIKELY)
  ((($DOSE |Implant[5]|) VERY-LOW) -->
   ((CONCENTRATION
     (@ (LAYER (ANY-MEMBER (NEW-LAYER N-CHANNEL-LAYER |Implant[5]|)))
        (END-OF |Implant[5]|)))
    VERY-LOW)
   VERY-LIKELY)
  ((($DOSE |Implant[5]|) VERY-HIGH) -->
   ((CONCENTRATION
     (@ (LAYER (ANY-MEMBER (NEW-LAYER N-CHANNEL-LAYER |Implant[5]|)))
        (END-OF |Implant[5]|)))
    VERY-HIGH)
   VERY-LIKELY)
  ((($DOSE |Implant[5]|) LOW) -->
   ((CONCENTRATION
     (@ (LAYER (ANY-MEMBER (NEW-LAYER N-CHANNEL-LAYER |Implant[5]|)))
        (END-OF |Implant[5]|)))
    LOW)
   VERY-LIKELY)
  ((($DOSE |Implant[5]|) HIGH) -->
   ((CONCENTRATION
     (@ (LAYER (ANY-MEMBER (NEW-LAYER N-CHANNEL-LAYER |Implant[5]|)))
        (END-OF |Implant[5]|)))
    HIGH)
   VERY-LIKELY))



Running rule INITIAL-DOPING-AFFECTS-FINAL -- Executed in 0.2 seconds
Running rule INITIAL-DOPING-AFFECTS-FINAL produces 4 rules:

((((START-CONCENTRATION SUBSTRATE) VERY-LOW) -->
   ((CONCENTRATION SUBSTRATE) VERY-LOW) VERY-LIKELY)
  (((START-CONCENTRATION SUBSTRATE) VERY-HIGH) -->
   ((CONCENTRATION SUBSTRATE) VERY-HIGH) VERY-LIKELY)
  (((START-CONCENTRATION SUBSTRATE) LOW) -->
   ((CONCENTRATION SUBSTRATE) LOW) VERY-LIKELY)
  (((START-CONCENTRATION SUBSTRATE) HIGH) -->
   ((CONCENTRATION SUBSTRATE) HIGH) VERY-LIKELY))
```

## Category 2 Causal Associations

```
Running rule INITIAL-DOPANT-TYPE-AFFECTS-FINAL -- Executed in 0.1
seconds
Running rule INITIAL-DOPANT-TYPE-AFFECTS-FINAL produces 1 rule:

((((START-DOPANT-TYPE SUBSTRATE) WRONG-TYPE) -->
  ((DOPANT-TYPE SUBSTRATE) WRONG-TYPE) VERY-LIKELY))



Running rule OXIDATION-STEP-OMITTED -- Executed in 0.5 seconds
Running rule OXIDATION-STEP-OMITTED produces 2 rules:

((|Oxidation[1]| NOT-DONE) --> (LAYER-7 NON-EXISTENT) VERY-LIKELY)
((|Oxidation[1]| NOT-DONE) --> (LAYER-8 NON-EXISTENT) VERY-LIKELY)



Running rule IMPLANT-STEP-OMITTED -- Executed in 0.2 seconds
Running rule IMPLANT-STEP-OMITTED produces 10 rules:

((((PARAMETER $DOPANT |Implant[1]|) WRONG-TYPE) -->
  ((@ (LAYER (ANY-MEMBER (NEW-LAYER SUBSTRATE |Implant[1]|)))
      (END-OF |Implant[1]|))
   NON-EXISTENT)
  MAYBE)
 ((|Implant[1]| NOT-DONE) -->
  ((@ (LAYER (ANY-MEMBER (NEW-LAYER SUBSTRATE |Implant[1]|)))
      (END-OF |Implant[1]|))
   NON-EXISTENT)
  VERY-LIKELY)
 (((PARAMETER $DOPANT |Implant[4]|) WRONG-TYPE) -->
  ((@ (LAYER (ANY-MEMBER (NEW-LAYER P-CHANNEL-LAYER |Implant[4]|)))
      (END-OF |Implant[4]|))
   NON-EXISTENT)
  MAYBE)
 ((|Implant[4]| NOT-DONE) -->
  ((@ (LAYER (ANY-MEMBER (NEW-LAYER P-CHANNEL-LAYER |Implant[4]|)))
      (END-OF |Implant[4]|))
   NON-EXISTENT)
  VERY-LIKELY)
 (((PARAMETER $DOPANT |Implant[4]|) WRONG-TYPE) -->
  ((@ (LAYER (ANY-MEMBER (NEW-LAYER POLYSILICON-LAYER |Implant[4]|)))
      (END-OF |Implant[4]|))
   NON-EXISTENT)
  MAYBE)
 ((|Implant[4]| NOT-DONE) -->
  ((@ (LAYER (ANY-MEMBER (NEW-LAYER POLYSILICON-LAYER |Implant[4]|)))
      (END-OF |Implant[4]|))
   NON-EXISTENT)
  VERY-LIKELY)
 (((PARAMETER $DOPANT |Implant[5]|) WRONG-TYPE) -->
  ((@ (LAYER (ANY-MEMBER (NEW-LAYER N-CHANNEL-LAYER |Implant[5]|)))
      (END-OF |Implant[5]|))
   NON-EXISTENT)
  MAYBE)
 ((|Implant[5]| NOT-DONE) -->
  ((@ (LAYER (ANY-MEMBER (NEW-LAYER N-CHANNEL-LAYER |Implant[5]|)))
```

```
      (END-OF |Implant[5]|))
   NON-EXISTENT)
  VERY-LIKELY)
 (((PARAMETER $DOPANT |Implant[5]|) WRONG-TYPE) -->
  ((@ (LAYER (ANY-MEMBER (NEW-LAYER POLYSILICON-LAYER |Implant[5]|)))
      (END-OF |Implant[5]|))
   NON-EXISTENT)
  MAYBE)
 ((|Implant[5]| NOT-DONE) -->
  ((@ (LAYER (ANY-MEMBER (NEW-LAYER POLYSILICON-LAYER |Implant[5]|)))
      (END-OF |Implant[5]|))
   NON-EXISTENT)
  VERY-LIKELY))
```

```
Running rule IMPLANT-NOT-MASKED-1 -- Executed in 0.4 seconds
Running rule IMPLANT-NOT-MASKED-1 produces 5 rules:

(((|Implant[1]| NOT-MASKED) -->
  ((DOPANT-TYPE SUBSTRATE) WRONG-TYPE) MAYBE)
 ((|Implant[4]| NOT-MASKED) -->
  ((DOPANT-TYPE P-CHANNEL-LAYER) WRONG-TYPE) MAYBE)
 ((|Implant[4]| NOT-MASKED) -->
  ((DOPANT-TYPE POLYSILICON-LAYER) WRONG-TYPE) MAYBE)
 ((|Implant[5]| NOT-MASKED) -->
  ((DOPANT-TYPE N-CHANNEL-LAYER) WRONG-TYPE) MAYBE)
 ((|Implant[5]| NOT-MASKED) -->
  ((DOPANT-TYPE POLYSILICON-LAYER) WRONG-TYPE) MAYBE))
```

## Category 3 Causal Associations

```
Running rule STRUCTURE-MASKED-IMPLANT -- Executed in 1.0 seconds
Running rule STRUCTURE-MASKED-IMPLANT produces 4 rules:

((((WIDTH POLYSILICON-LAYER IN WAFER-REGION-7) LOW) -->
  ((WIDTH P-CHANNEL-LAYER IN WAFER-REGION-7) LOW) VERY-LIKELY)
 (((WIDTH POLYSILICON-LAYER IN WAFER-REGION-7) HIGH) -->
  ((WIDTH P-CHANNEL-LAYER IN WAFER-REGION-7) HIGH) VERY-LIKELY)
 (((WIDTH POLYSILICON-LAYER IN WAFER-REGION-4) LOW) -->
  ((WIDTH N-CHANNEL-LAYER IN WAFER-REGION-4) LOW) VERY-LIKELY)
 (((WIDTH POLYSILICON-LAYER IN WAFER-REGION-4) HIGH) -->
  ((WIDTH N-CHANNEL-LAYER IN WAFER-REGION-4) HIGH) VERY-LIKELY))
```

```
Running rule INSUFFICIENT/EXCESS-LATERAL-ETCH -- Executed in 0.3 seconds
Running rule INSUFFICIENT/EXCESS-LATERAL-ETCH produces 8 rules:

(((|Etch[1]| OVER-ETCH) -->
  ((WIDTH POLYSILICON-LAYER IN WAFER-REGION-7) VERY-LOW) VERY-LIKELY)
 ((|Etch[1]| OVER-ETCH) -->
  ((WIDTH POLYSILICON-LAYER IN WAFER-REGION-7) LOW) MAYBE)
 ((|Etch[1]| UNDER-ETCH) -->
  ((WIDTH POLYSILICON-LAYER IN WAFER-REGION-7) VERY-HIGH) VERY-LIKELY)
 ((|Etch[1]| UNDER-ETCH) -->
  ((WIDTH POLYSILICON-LAYER IN WAFER-REGION-7) HIGH) MAYBE)
 ((|Etch[1]| OVER-ETCH) -->
  ((WIDTH POLYSILICON-LAYER IN WAFER-REGION-4) VERY-LOW) VERY-LIKELY)
 ((|Etch[1]| OVER-ETCH) -->
  ((WIDTH POLYSILICON-LAYER IN WAFER-REGION-4) LOW) MAYBE)
 ((|Etch[1]| UNDER-ETCH) -->
  ((WIDTH POLYSILICON-LAYER IN WAFER-REGION-4) VERY-HIGH) VERY-LIKELY)
 ((|Etch[1]| UNDER-ETCH) -->
  ((WIDTH POLYSILICON-LAYER IN WAFER-REGION-4) HIGH) MAYBE))



Running rule INSUFFICIENT/EXCESS-PHOTORESIST-EXPOSURE -- Executed in 1.2
seconds
Running rule INSUFFICIENT/EXCESS-PHOTORESIST-EXPOSURE produces 8 rules:

(((|"Select Mask"[3]| OVER-EXPOSE) -->
  ((WIDTH PHOTORESIST IN WAFER-REGION-4) VERY-LOW) VERY-LIKELY)
 ((|"Select Mask"[3]| OVER-EXPOSE) -->
  ((WIDTH PHOTORESIST IN WAFER-REGION-4) LOW) MAYBE)
 ((|"Select Mask"[3]| UNDER-EXPOSE) -->
  ((WIDTH PHOTORESIST IN WAFER-REGION-4) VERY-HIGH) VERY-LIKELY)
 ((|"Select Mask"[3]| UNDER-EXPOSE) -->
  ((WIDTH PHOTORESIST IN WAFER-REGION-4) HIGH) MAYBE)
 ((|"Select Mask"[3]| OVER-EXPOSE) -->
  ((WIDTH PHOTORESIST IN WAFER-REGION-7) VERY-LOW) VERY-LIKELY)
 ((|"Select Mask"[3]| OVER-EXPOSE) -->
  ((WIDTH PHOTORESIST IN WAFER-REGION-7) LOW) MAYBE)
 ((|"Select Mask"[3]| UNDER-EXPOSE) -->
  ((WIDTH PHOTORESIST IN WAFER-REGION-7) VERY-HIGH) VERY-LIKELY)
 ((|"Select Mask"[3]| UNDER-EXPOSE) -->
  ((WIDTH PHOTORESIST IN WAFER-REGION-7) HIGH) MAYBE))
```

```
Running rule RESIST-MASKED-ETCH -- Executed in 0.3 seconds
Running rule RESIST-MASKED-ETCH produces 8 rules:

((((WIDTH PHOTORESIST IN WAFER-REGION-7) VERY-LOW) -->
  ((WIDTH POLYSILICON-LAYER IN WAFER-REGION-7) VERY-LOW) MUST)
 (((WIDTH PHOTORESIST IN WAFER-REGION-7) VERY-HIGH) -->
  ((WIDTH POLYSILICON-LAYER IN WAFER-REGION-7) VERY-HIGH) MUST)
 (((WIDTH PHOTORESIST IN WAFER-REGION-7) LOW) -->
  ((WIDTH POLYSILICON-LAYER IN WAFER-REGION-7) LOW) MUST)
 (((WIDTH PHOTORESIST IN WAFER-REGION-7) HIGH) -->
  ((WIDTH POLYSILICON-LAYER IN WAFER-REGION-7) HIGH) MUST)
 (((WIDTH PHOTORESIST IN WAFER-REGION-4) VERY-LOW) -->
  ((WIDTH POLYSILICON-LAYER IN WAFER-REGION-4) VERY-LOW) MUST)
 (((WIDTH PHOTORESIST IN WAFER-REGION-4) VERY-HIGH) -->
  ((WIDTH POLYSILICON-LAYER IN WAFER-REGION-4) VERY-HIGH) MUST)
 (((WIDTH PHOTORESIST IN WAFER-REGION-4) LOW) -->
  ((WIDTH POLYSILICON-LAYER IN WAFER-REGION-4) LOW) MUST)
 (((WIDTH PHOTORESIST IN WAFER-REGION-4) HIGH) -->
  ((WIDTH POLYSILICON-LAYER IN WAFER-REGION-4) HIGH) MUST))



Running rule THERMAL-OXIDATION-REDISTRIBUTES-IMPURITIES-N -- Executed in
0.5 seconds
Running rule THERMAL-OXIDATION-REDISTRIBUTES-IMPURITIES-N produces 4
rules:

((((PARAMETER $DURATION |Oxidation[1]|) VERY-SHORT) -->
  ((CONCENTRATION LAYER-4) LOW) VERY-LIKELY)
 (((PARAMETER $DURATION |Oxidation[1]|) SHORT) -->
  ((CONCENTRATION LAYER-4) LOW) VERY-LIKELY)
 (((PARAMETER $DURATION |Oxidation[1]|) VERY-LONG) -->
  ((CONCENTRATION LAYER-4) HIGH) LIKELY)
 (((PARAMETER $DURATION |Oxidation[1]|) LONG) -->
  ((CONCENTRATION LAYER-4) HIGH) VERY-LIKELY))



Running rule THERMAL-OXIDATION-REDISTRIBUTES-IMPURITIES-P -- Executed in
0.2 seconds
Running rule THERMAL-OXIDATION-REDISTRIBUTES-IMPURITIES-P produces 4
rules:

((((PARAMETER $DURATION |Oxidation[1]|) VERY-SHORT) -->
  ((CONCENTRATION LAYER-6) HIGH) VERY-LIKELY)
 (((PARAMETER $DURATION |Oxidation[1]|) SHORT) -->
  ((CONCENTRATION LAYER-6) HIGH) VERY-LIKELY)
 (((PARAMETER $DURATION |Oxidation[1]|) VERY-LONG) -->
  ((CONCENTRATION LAYER-6) LOW) LIKELY)
 (((PARAMETER $DURATION |Oxidation[1]|) LONG) -->
 ((CONCENTRATION LAYER-6) LOW)
  VERY-LIKELY))



Running 15 generic rules produces 122 instantiations.
```