On the Complexity of the Design Synthesis Problem

Oded Maimon and Dan Braha

Abstract—**In this paper we present and analyze a formal model of a design process, emphasizing the synthesis part. The design artifact description is identified as an algebraic structure. The desired function and constraints are mapped to the artifact description using an evolutionary process that can be visualized as a feedback loop of analysis, synthesis and evaluation. A special case of the synthesis activity, called the Basic Synthesis Problem (BSP), is addressed. The BSP is shown to be NP-Complete. As a consequence, tractability can be obtained by enforcing constraints on the artifact structure. As such, we present a model having an element of descriptive design theory that is also a framework for the future development of computational support systems and automatic design tools.**

I. Introduction

A. The Analysis-Synthesis-Evaluation (ASE) Design Paradigm

Design can be viewed as the process of developing a solution to required specifications that performs some set of desired functions and, with some degree of optimality, meets a set of resource usage objectives. It involves wide use of domain specific knowledge and considerable problem solving skills to come up with clear, concise and unambiguous specifications of the designed artifact [9], [11]. There is no single model that can furnish a perfect definition of the design process. However, the Analysis-Synthesis-Evaluation (ASE) paradigm represents a widely held view in such diverse fields as structural engineering [23], device design [20], and software engineering [3], [21].

According to ASE, the design is thought to consist of three logically and temporally distinct stages: a stage of analysis of the requirements, then a stage of synthesis, followed by a stage of evaluation [2], [7], [14], [23]. Analysis is described as involving the identification of all possible factors that may be relevant to the given design situation, determination and resolution of all interactions among the factors and the eventual reduction of these factors to a complete set of specifications. Synthesis involves the construction of partial solutions for each of the distinct specifications and the integration of these partial designs into a complete form. Finally, the evaluation phase is concerned testing the design produced by the synthesis phase against the specifications identified during the analysis phase. In the event alternative forms were produced during synthesis, this is also the stage in which a choice is made between the alternatives as a result of evaluation. Several instances of these three phases may be required in order to progress from a more abstract level to a more concrete level. The ASE model of design process is inherently iterative; the designer repeatedly goes back to refine and improve her design until the design satisfies the requirements.

B. Scope and Organization of the Paper

This paper develops a generic formalism that aims to explain how design artifacts are represented, and how design processes conceptually perform in terms of knowledge manipulation (as captured by the ASE-based design paradigm). In this study, the term "design" is defined as a process. Given a description of a desired function and constraints, called specifications, provide a representation of an artifact that produces the function and satisfies the constraints. This representation, called artifact description or artifact structure, is identified as an algebraic structure. The mapping from functional requirements to artifact structure is identified as an evolutionary process consisting of similar activities (modeled through finite-state automata) repeated in a cyclic fashion. We address here a special case of the synthesis activity called the Basic Synthesis Problem (BSP). The term "basic synthesis" used in this study is defined as the complete specifications of 'primitive' components and their relations so as to meet a set of specifications of satisfactory performance. We show that the decision problem concerning the existence of a 'satisfying' artifact is NP-Complete. By enforcing certain constraints on the artifact structure, we obtain upper-bounds on the number of possible design solutions. These bounds are instrumental for devising a heuristic strategy for the BSP. Thus our results are guidelines for developing algorithms to search for optimal and suboptimal design solutions.

The rest of the paper is organized as follows: Section II provides two scenarios of real engineering design problems, given to gain some insight into the design process and to motivate the proposed model. The design process model is presented in Section III. Section IV addresses the BSP and proves its computational intractability. The consequences are then explored. In Section V, we examine the class of Constrained Basic Synthesis Problems (CBSP). We use an information-theoretic approach to derive a universal upper bound on the number of possible design solutions. In Section VI, we derive a refined upper bound on the number of possible design solutions, assuming a probabilistic search strategy for solving the CBSP. Section VII concludes the paper.

II. Two Scenarios of ASE-Based Design Methods

Before we discuss more closely the ASE model and its computational aspects, it is useful to consider some specific instances of design processes that appear to follow the paradigm. These examples will also serve to illuminate certain points of discussion later in the paper.

A. Mechanical Fasteners Design

The design of mechanical fasteners is a common design problem in the realm of mechanical engineering. The function of a fastener is to hold two or more parts together. There are numerous types of existing mechanical fasteners (see Fig. 1). The proposed problem is to design a new fastener according to certain given specifications based on the knowledge that we already have from the existing fasteners. At the lowest level, a fastener might be described in terms such as head radius, thread pitch, and thread depth (assuming that it has threads). Functional requirements, however, usually do not come so neatly packaged. Instead, the requirements usually start out at a high level (such as the strength or precision of the fastener). From these higher-level requirements, a more detailed description evolves through the design process.

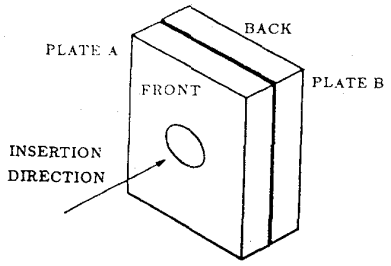
1) The Knowledge-Base: The first step is to build a data base of this knowledge. The existing designs are represented in a set of design descriptions that include structural, functional and the causal relationships between function and structure. The structural

Manuscript received May 7, 1993; revised January 19, 1995.

O. Maimon is with the Department of Industrial Engineering, Tel-Aviv University, Ramat-Aviv, Tel-Aviv 69978, Israel.

D. Braha is with the Department of Manufacturing Engineering, Boston University, Boston, MA 02215 USA.

Publisher Item Identifier S 1083-4427(96)00063-X.



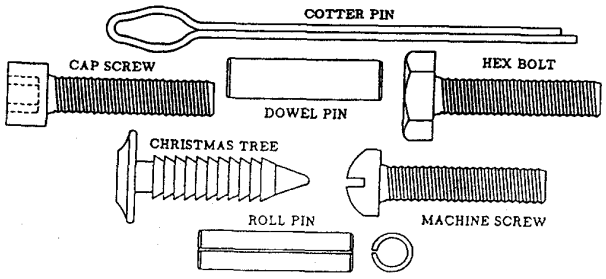


Fig. 1. Fasteners in the knowledge base.

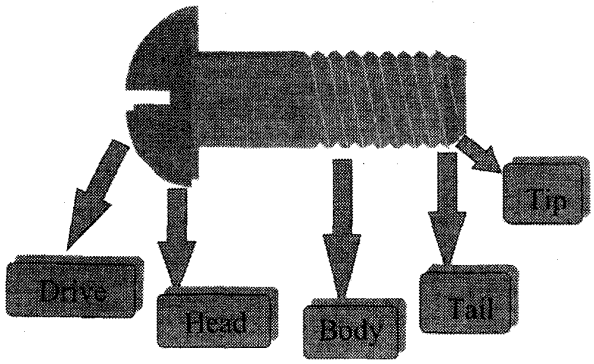


Fig. 2. Hierarchy structure of a fastener.

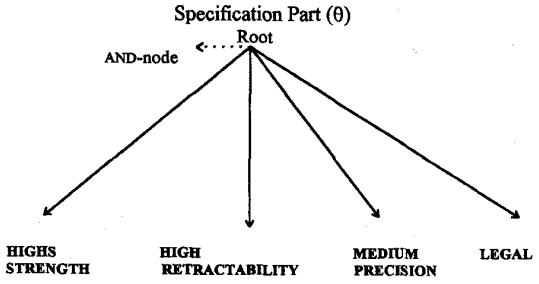


Fig. 3. Synthesis state 0.

description of the device and its components are listed. In the case of fasteners, the general hierarchy is composed of the five components that are common to all fasteners; drive, head, body, tail and a tip as shown in Fig. 2.

The behavioral abstraction of a design is usually determined by the structural hierarchy. For instance, the function of a fastener is to hold two or more parts together. This is done by the fastener which transmits the load to the fastened parts. The relation is usually in the form of a causal one governed by the physical laws. The behavior of the components follows the same principle. For an illustration of the relationships; the strength of the fastener is determined by the static fatigue, stress rupture, and impact strength. The fatigue is determined by the material, head size, thread size, threaded length, fillet, fabrication, and surface structure. Those properties, in turn, are determined by the physical attributes of the components and entity. The same would apply for the rest of the properties of the fastener. This would be captured and represented as the causal and factual knowledge of the system in a causal network and as an if-then structure. These structures would link the functions to the attributes and the attributes to the components.

2) The Task: To perform the synthesis task, the system is designed to transform functional specifications to a structural descriptions. A typical input to the design system is a conjunction of functional attributes. The program task is to propose a fastener design which will meet a particular set of functional requirements. The result of applying the above design procedure to an input specification is a set of possible solutions, each of which consists of a set of five structural attributes corresponding to the five parts of the fastener structure.

3) A Run Time Design Process: For our fastener design, the initial functional requirements come down as. The 'LEGAL' requirement indicates that the design solution should be given in terms of its five structural attributes (corresponding to the five parts of the fastener structure; i.e., drive, head, body, tail and a tip). The first iteration may be to realize that strength is characterized separately by the head and tail of the fastener. By applying the knowledge-base of fasteners and analyzing the current requirements, it is concluded that both of them demand high strength. The second iteration may be to get a better definition of retractibility. There is a choice of how fasteners are driven and that in order to get high retractibility we need rotary-mode fasteners. In the next iteration, we realize that medium precision translates into a looser body fit. At some point, synthesis decisions begin to be made as the final configuration of the design begins to take place. From the current information, we infer that the head of the fastener needs a shoulder. And that the tail of the fastener will be fastened using threads. And that the drive type will be a hex head. And that the body will have threads. In the next synthesis state, it is found that the threads-type tail of the fastener derives the requirement of TAIL-MODE-ROTARY. Since this structural attribute is already fixed, the design part and the specification part remain unchanged. Finally, by testing the tentative part against the specification part it is concluded that all specifications are satisfied except for the last requirement (LEGAL). A misfit still remains since the tip part was not determined as yet. The transition strategy is to determine an attribute for the tip part, which is not in conflict with the already satisfied specifications. Hence, it was decided that the tip of the fastener should be chamfered in order to maximize reversibility. This process can be continued until the physical attributes of the final fastener are discovered (see Figs. 3–11).

B. A Supercritical Fluid Chromatography (SFC) Design [23]

SFC is a device which provides detailed analysis of a chemical mixture. A brief sketch of the evolution of the SFC is as follows: By studying the literature and understanding the chromatographic process (an analysis activity), designers arrive at a tentative design solution (a synthesis activity) which consists of pump, control and

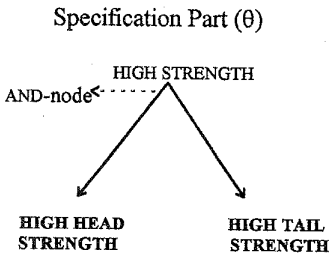


Fig. 4. Synthesis state 1 (partial).

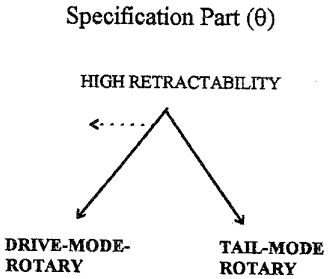


Fig. 5. Synthesis state 2 (partial).

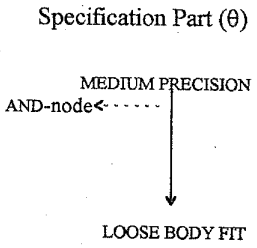


Fig. 6. Synthesis state 3 (partial).

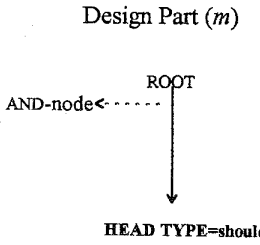


Fig. 7. Synthesis state 4 (partial).

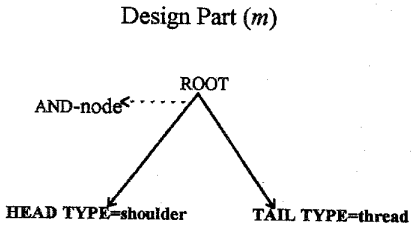


Fig. 8. Synthesis state 5 (partial).

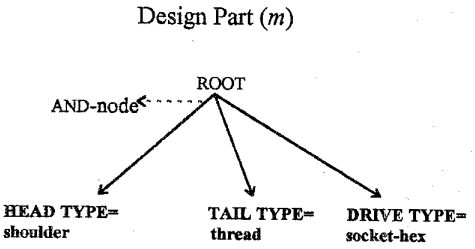


Fig. 9. Synthesis state 6 (partial).

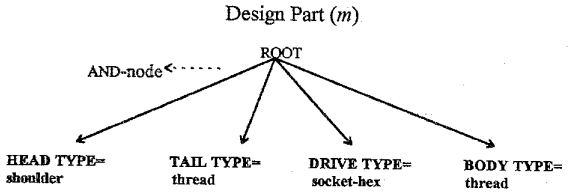


Fig. 10. Synthesis state 7 (partial).

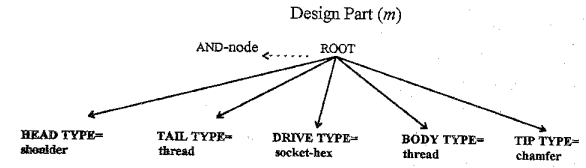


Fig. 11. Synthesis state 9 (partial).

oven units. The oven unit is further divided into injection, oven and detector units (a synthesis activity). This tentative design amounts to the following behavior: carbon dioxide is compressed to the desired pressure and pumped into the column and the detector, afterwards the sample mixture is injected through the injector and transported to the column by carbon dioxide where separation takes place. The detector controls the pump unit by detecting the amount of the various compounds of the sample mixture.

Let us examine the (partial) evolution of the design pump unit: The initial design specifications were stated as: 1) it has to be pulseless, 2) it should be able to compress carbon dioxide. By invoking a large knowledge-base and analyzing the initial design specifications, the designer came up with a syringe pump. After choosing the type of the pump, an additional specification derived was: 3) the number of experiments with a single filling should be optimized. This specification was modified to the following specification: 4) the pump must be large. Based on previous knowledge, the designer refined the specifications regarding the pump to include the design of its subunits, involving the power source, cylinder, gear, stepper motor, etc.

III. The Model

A complete modeling of the design process must answer three fundamental questions: how is the entity (artifact) representation and knowledge structure generated, how are the specifications (a set of constraints or requirements) represented, and how is the process (the mapping from the requirements to the artifact structures) represented and executed? In this section we present only the essentials for the understanding of the forthcoming discussion on the characterization of the synthesis activity. For a detailed definition of the design process see [19].

Definition 1 (Design Process): The design process model is denoted by the tuple $DP = \langle D, L, A, S, E \rangle$. D specifies the *artifactspace*, which consists of various components, some of which are generated as part of the solution description. L is the set of explicit constraints (or *specifications*) posed on the problem. The *analyzer* (A) and the *synthesizer* (S) are transformation operators that utilize domain knowledge and generate new sets of—possible abstract—specifications and solutions, respectively. The *evaluator* (E) evaluates the current design produced by the synthesis step by testing it against the current specifications.

A. Artifact Space (D)

It is obvious that the ability to analyze, model, and describe design artifacts in a precise, formal and unambiguous manner is not merely desirable but imperative. For only then can one attempt to test a design against the requirements and determine unambiguously whether or not the former satisfies the latter. Therefore, the artifact space is based on the postulate that any knowledge representation is built upon the multiplicity of objects (henceforth *modules*) and relationships (henceforth *relations*) among them. Consequently, the design artifact is represented by a pair $\langle M, C \rangle$. M stands for the set of modules which the artifact is comprised of; and C denotes the set of relations that represent the relationships among the modules. In order to capture the essence of design, a hierarchical construction of systems from subsystems is also developed. Consequently, the universal set of modules is classified into *basic* and *complex modules*. Basic modules represent entities that can not be defined in terms of others. Complex modules are defined hierarchically in terms of others modules, where the interaction effects are captured.

Definition 2 (Artifact Space): The artifact space is denoted by the tuple $\langle M^0, C^0, M^* \rangle$. Let us discuss each component in turn:

Primitive Modules: M^0 denotes the set of primitive modules. Primitive modules can not be defined in terms of other modules (except trivially by themselves). The set M^0 represents primitive components which can be assembled to construct complex modules. Consider for example the design of a computer environment (e.g., communication networks): The basic modules can be programs, data files, subroutines. In the design of an analog circuit primitive modules represent resistors, capacitors, operational amplifiers, diodes.

Relations: C^0 denotes a preassigned universal set of relations. $c \in C^0$ denotes a collection of tuples, each representing a relation among the elements within the tuple. The relationships among modules may also be expressed in nonnumerical terms (e.g., using symbolic logic). Consider for example the design of an analog circuit. The relations among primitive modules may be represented by 'information' and 'physical' relations among capacitors, resistors etc.

Complex Modules: M^* is a universal set of modules, e.g., $M^* = \{\text{Primitive modules}\} \cup \{\text{Complex Modules}\}$, where a complex module is defined hierarchically in terms of previously-described modules. Since this definition is hierarchical, the terms *'artifact,' 'complex module'* and *'module'* are used interchangeably. Let an artifact, $m \in M^*$, be defined by a set $C \subseteq C^0$, such that $C = \{c_i\}_{i \in I}$, where $c_i = \{M_{i,k} = \langle m_{i1}, m_{i2}, \ldots, m_{in} \rangle_k : c_i(\langle m_{i1}, m_{i2}, \ldots, m_{in} \rangle_k)$ is true$\} \subseteq (M^*)^n$; with $c_i(\langle m_{i1}, m_{i2}, \ldots, m_{in} \rangle_k)$ being a formula associated with the relation c_i, and $(M^*)^n$ is a Cartesian product of n copies of the set M^*. The kth element $M_{i,k}$ of c_i, is termed as an assignment of the relation c_i.

Hence, a system m is defined in terms of a set $\{M_{i,k}\}$ and a set C, such that $C \subseteq C^0$; and $M_{i,k}$ denotes an ordered set of modules. Denote $M = \bigcup_{i,k} M_{i,k}$ as the set of labels that can be assigned to the relation set C. For convenience the notation $m = \langle M, C \rangle$ is often used. The set of artifacts that can be described over a set of labels M and a set of relations C is denoted as $\overline{\langle M, C \rangle}$.

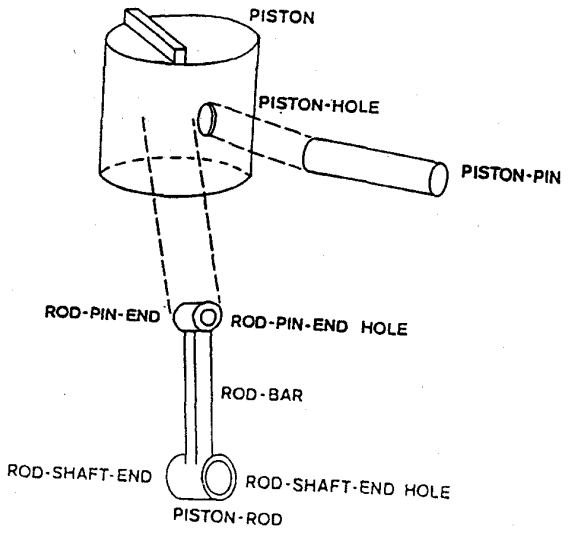


Fig. 12. The piston subassembly of a model aircraft engine.

Example 1 (Piston Subassembly): That any knowledge representation is built upon the multiplicity of objects and relationships among them is an empirical proposition for which evidence has been provided in [4], [17]–[19]. Unfortunately, space limitations confine us to only a few examples in this report. Here we provide further evidence corroborating the proposition by demonstrating how traditional computer models of geometrically complex objects conform to the entity-relational knowledge representation postulate.

The problem of building computer models of geometrically complex objects has been addressed in the context of graphics [5], computer aided design [25] and mechanical assembly [16]. Two major methods have been devised. The *surface* approach [1] describes the surfaces of the objects by specifying the vertices and edges of the faces of polyhedra or, for curved objects, the crossections or surface patches. The *solid* approach [5] approximate complex objects by composing several simpler volumes. There also exist some hybrid systems that allow both types of descriptions.

The Artifact Description: Fig. 12 shows a piston subassembly from a model aircraft engine. Fig. 13 shows a schematic description of the parts in the piston component subassembly. The parts are arranged hierarchically, where any desired subparts can be represented as nodes in the part model trees. Each node has information regarding the *size*, *type* and *relative position* of the subparts. All the subparts, including holes, are approximated as *rectangular* or *octagonal right prisms*. This provides a uniform internal representation for all the object types. This representation simplifies the definitions of the spatial modeling operations. By generalizing to polyhedra we could approximate the desired volumes to any required accuracy.

The Formal Representation Scheme.

Primitive Modules:

- Polyhedral solids whose crossections are regular polygons. For example: 1) rectangular solid, and 2) octagonal solid which is meant to approximate a cylinder.
- Properties, attributed to primitive objects, that specify their size parameters, vertex points, equations for the planes of the faces, generalized position and orientation, etc.

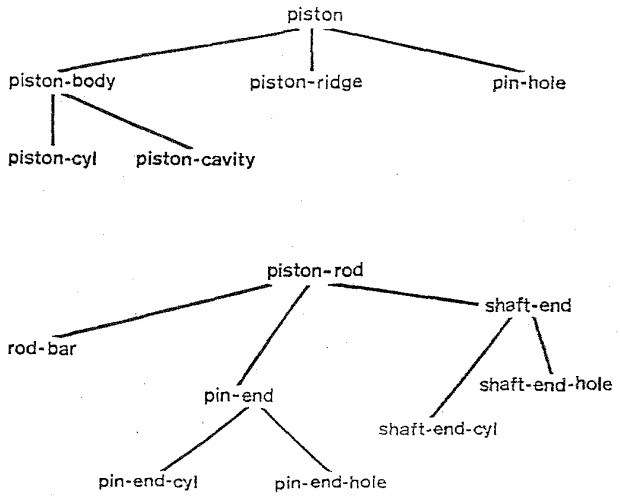


Fig. 13. The tree-structured relationship of parts in the piston subassembly.

Complex Modules: Complex modules are represented as unions of other objects (primitive as well as complex). Holes and cut-outs can be treated uniformly as objects by allowing primitive objects to have negative volumes. For example, the cavity of the piston, shown in Fig. 12, can be represented as two cylindrical hole and a cuboid to approximate its elliptical crossection.

Relations: The set of relations includes: 'TYPE' denoting the Polyhedral solid that approximate the object; 'X', 'Y' and 'Z' denoting the position of the object. 'LINK' indicating the coordinate transformation between the local coordinate systems of two objects. Similarly we have 'LENGTH,' 'NAME,' 'RADIUS,' 'OFFSET' and 'ANGLES'.

Modeling the Piston Rod. Let us present the model of the piston rod for the model aircraft engine (see Fig. 12). First we define the components parts of the object. The components parts are: piston-rod's shaft, pin-end, pin-end's hole, shaft-end, and shaft-end's hole. Formally:

BAR (TYPE(rect), $X(0.2)$, $Y(0.2)$, $Z(0.62)$)
SHAFT-END-CYL (TYPE(cyl), RADIUS(0.156), LENGTH(0.114))
SHAFT-END-HOLE (TYPE(cyl-hole), RADIUS(0.089), LENGTH(0.114))
PIN-END-CYL (TYPE(cyl), RADIUS(0.134), LENGTH(0.16))
PIN-END-HOLE (TYPE(cyl-hole), RADIUS(0.081), LENGTH(0.16))

Note that the foregoing objects define complex modules of order 2. For example, 'rect' is a primitive module, 'TYPE(rect)' is complex module of order 1, and BAR(...) is a complex module of order 2.

The next step is to indicate the relationships between the various parts. The simplest links in the model of the piston rod are the relationships of the holes to their corresponding cylinders since they are aligned and concentric:

LINK (SHAFT-END-HOLE, SHAFT-END-CYL)
LINK (PIN-END-HOLE, PIN-END-CYL)

Then we place the hollow cylinders at the ends of the bar:

LINK (SHAFT-END-CYL, BAR)
LINK (PIN-END-CYL, BAR)

Finally, we need also to denote the position and orientation of either one of the hollow cylinders relative to the BAR:

OFFSET (SHAFT-END-CYL, BAR, $X(0)$, $Y(0)$, $Z(0.466)$)
ANGLES (SHAFT-END-CYL, BAR, $X(0)$, $Y(p/2)$, $Z(0)$)
OFFSET (PIN-END-CYL, BAR, $X(0)$, $Y(0)$, $Z(-0.444)$)
ANGLES (PIN-END-CYL, BAR, $X(0)$, $Y(p/2)$, $Z(0)$)

B. Specifications (L)

It is imperative to describe the specifications in a precise, formal and unambiguous manner. For only then can one attempt to test a design against the requirements and determine unambiguously whether or not the former satisfies the latter. Ideally, the specifications are described in the designer's natural language. In [19] both propositional and first-order logic specification language were developed. In this paper, we refer to the specifications indirectly. Therefore, it was deemed not to present the formal model.

C. Transformation Operators (A, S & E)

You are given a set of requirements and a blank piece of paper. How do you proceed? We view the design process as a goal-directed derivation process which starts with an initial set of design specifications and terminates with one or more artifacts. By adaptively modifying pairs of design and specifications, from one cycle (step) to the next, we arrive at a design solution. In particular, the following basic terminology is used: A synthesis state is described by:

1) *Design part*: the tentative design solution (artifact) synthesized up to this point of the execution.
2) *Specification part*: the set of requirements that remains to be satisfied.

Initially the design part is empty and the specification part includes the initial requirements. Let us denote the synthesis state by a pair $\langle m, \theta \rangle$, where m is the design part and θ is the specification part. A design part may be said to represent *first order constraints*; features that serve as (or as a part of) the blueprint for the artifact's implementation. The specification part may be said to form *higher order constraints*; properties, assertions or predicates that are satisfied by the lower order constraints (e.g., functional, performance, reliability, manufacturability and aesthetic constraints). Although the distinction between "design" and "specifications" is quite frequently blurred and meaningless, we will make the distinction because of methodological arguments and mathematical convenience.

A *synthesis cycle* (step) corresponds to a transition (transformation) from one synthesis state to another synthesis state and is featured by:

Evaluation: testing and verifying the current design part against the available specifications. Testing the tentative design part against the specification part involves a wide range of reasoning types, such as classical and approximate logical systems (theorem provers, qualitative reasoning); experiments (simulations); knowledge accumulated from previous experience; and common sense reasoning (heuristics and 'rules of thumb').

Analysis & Synthesis: Modifying the synthesis state, resulting in a new pair of (design, specifications). The modification can be directed in two manners, synthesizing a new design part or identifying all possible factors that may be relevant to the current synthesis stage and the eventual reduction (analysis) of these factors to a new specification part. In [19], the analysis and synthesis transformations were modeled through finite state automata.

A synthesis state is *terminal* in either of two cases: 1) the specification part is *satisfiable* by the design part (in which case we have a solution), or 2) neither the design nor the specification parts can be further modified. The former is termed a *successful pair* whereas the latter is termed a *failed* pair.

An *execution* of a design process is simply a finite sequence of synthesis states, resulting in a terminal state.

The rest of the paper addresses and examines the computational complexity attached to a special case of the synthesis activity termed the *Basic Synthesis Problem* (BSP). Rigorous description of this problem will serve to illuminate the intractability properties of the design process.

IV. The Basic Synthesis Problem

A. Problem Formulation

The term 'basic synthesis' is defined as the complete specifications of primitive components and their relations so as to meet a set of specifications of satisfactory performance, which correctly implies the domain-independent nature of the design as a generic activity. Traditional engineering design methods prefer to use specifications of satisfactory performance over using optimal performance, since the designer is constantly faced with the problem of bounded rationality [22]. The model of bounded rationality takes as self-evident limitations on the cognitive and information processing capability of the designer's decision making.

In practice, designers often set some criteria of satisfactoriness and if the design meets the criteria, the design problem is considered to have been 'solved'. These criteria are mostly conflicting and heterogeneous in the same sense that the quality of the compared alternatives cannot be adequately expressed by a single integral criterion formed as a composition of the original (partial) criteria. Therefore, single-criterion optimization is often insufficient for choosing the best designs. The multiple criteria can include manufacturing and marketing considerations, in addition to traditional measures.

Example 2: Consider a multistorey reinforced concrete building design. A complete evaluation of a design solution (limited to the main beams) mainly involves the evaluation of the following four types of criteria [15].

- Slenderness of beams. This checks the adherence of the design to zoning regulations regarding span/depth restrictions of the beams.
- Interference. This is based on the clearance between floor and roof, which in turn, is decided by the deepest beam in the plan.
- Beam-column compatibility. At every junction of the beam and column, a check is made to see if the smaller of the two columns dimensions is the same as the breadth of the beam that frames into that side. The evaluation may be based on the percentage of all such instances that violate this requirement.
- Adjacency—The desirability of two adjacent beams having the same depth is sought for. The evaluation may be based on the percentage of all such instances that violate this requirement.

The multiple criteria optimization is modeled through the evaluation operator. The evaluation operator measures the degree of efficiency of an artifact, defined as the degree of closeness (the less the better) to the actual specifications desired. Formally,

Definition 3 (Evaluation Operator): A mapping $E_\theta : M^* \rightarrow \Re^n; \theta$ is an ordered set of specifications alternatives and $E_\theta(m) = (E_{\theta_1}(m), E_{\theta_2}(m), \ldots, E_{\theta_n}(m))$, such that $E_{\theta_i}(m)$ measures the degree of closeness of to the specification θ_i. E_θ induces a preference structure on the artifact space, i.e.,

1) $E_{\theta_i}(m_1) < E_{\theta_i}(m_2) \Leftrightarrow$ The designer strictly prefers module m_1 over module m_2, relative to the specification attribute θ_i.
2) $E_{\theta_i}(m_1) = E_{\theta_i}(m_2) \Leftrightarrow$ The designer is indifferent regarding choice of m_1 over module m_2, relative to the specification attribute θ_i.

The designer often defines, prior to solving the BSP, a *threshold* vector $\underline{K} = (k_1, k_2, \ldots, k_n)$. $\underline{K}$ represents (pointwise) the maximum degree of closeness (to θ), which is still accounted efficient. We term $\underline{K}$ as the designer's *aspiration level*. Having defined the aspiration level, the BSP is formulated as the decision problem concerning the existence of a module evaluated *below* the aspiration level. Formally,

Definition 4 (Basic Synthesis Problem): Given a set of modules M, a set of relations C and a positive vector $\underline{K} \in \Re^n$, are there subsets $M_0 \subseteq M, C_0 \subseteq C$ and module $m \in \overline{\langle M_0, C_0 \rangle}$ such that $E_\theta(m) \leq K$?

Special instances of the BSP include PCB's design ("packing", "placement" and "routing"); 0 logic gates circuit satisfiability; and certain graph enumeration and isomorphism problems in the realm of mechanisms design. Toward examining thoroughly the computational aspects of the BSP, let us first identify one problem instance of the BSP.

Example 3 (Minimizing Microinstruction Size) [8, 12, 13]:

Microprogrammed Control: Microprogramming is a technique for implementing the control function of a processor in a systematic and flexible manner. Every instruction in a CPU is implemented by a sequence of one or more sets of concurrent microinstraction. For example, a microinstruction represented by the symbol $\text{Reg}_1 \leftarrow \text{Reg}_2$, when executed by the control unit, causes the content of the specified register Reg_2 to be gated to the register Reg_1. Each microoperation is associated with a specific set of control lines which, when activated cause that microoepration to take place. Since the number of instructions and control lines is often in the hundreds, a *hardwired* control unit that select and sequences the control signals can be exceedingly complicated.

Microprogramming may be considered as an alternative to hardwired control circuits. The control signals to be activated at any time are specified by a word called a *microinstruction* which is fetched from a *control memory* COM in much the same way an instruction is fetched from the main memory. A set of related microinstructions is called a *microprogram.*

Parallelism in Microinstructions: Microoperation length is determined, at large, by the maximum number of simultaneous microoperations that must be specified. Therefore, microinstructions are often designed to take advantage of the fact that at the microprogramming level, many microoperations can be performed in parallel. For example, if microoperation m_1 writes into a register/store which is read by m_2 than m_1 and m_2 cannot be executed in parallel. Therefore, it is useful to divide the microoperation specification part of the microinstruction into k disjoint parts called *control fields.* Each control field encodes or represents a set of microoperations, any one of each can be executed concurrently with the microinstructions specified by the remaining control fields.

Minimally Encoded Microinstruction Organization (MEMO): Let us examine the problem of encoding the control fields such that the total number of bits in the control fields is a minimum [8]. Let $I_1, I_2, \ldots, I_m$ be a set of microinstructions for the computer that is being designed. Each microinstruction specifies a subset of the available microinstructions $S = \{S_1, S_2, \ldots, S_n\}$ which must be activated. An encoded control field can activate only one microoperation at a time. Two microoperations S_1 and S_2 can be included in the same control field only if they cannot be executed in parallel from the same microinstruction. Call S_1 and S_2 a *compatible pair.* A *compatible class* H_1 is a set of microoperations that are pairwise compatible. Each H_1 can, then, be encoded in a single field of the microinstructions using $B_i = \lceil \log_2 |H_i| + 1 \rceil$ bits. The total length of the microinstruction would be $B = \sum_{i=1}^{k} B_i$ bits.

The problem is, to determine a set $H_{\min}$ of compatible sets such that the corresponding length B, is the minimum. This problem is formulated in a BSP form as follows:

- The primitive modules (M) are identified with the set of microinstructions S;
- There are n types of relations (C), such that $c_j = \{\langle S_{i_1}, S_{i_2}, \ldots, S_{i_j} \rangle : \{S_{i_1}, S_{i_2}, \ldots, S_{i_j}\}$ is a compatible class that includes j microoperations$\}$.

- A solution for the BSP is specified in terms of a set of relations $\{c_i\}$.
- The evaluation operator (E) is the total length (in bits) of the microinstruction that corresponds to the BSP solution.

Special Case: Let $S = \{s_1, s_2, \ldots, s_8\}$ be the set of primitive modules. Consider the following pairs of microoperations that can be executed concurrently without any conflict in their resource usage: $\langle s_1, s_2\rangle$, $\langle s_1, s_3\rangle$, $\langle s_2, s_4\rangle$, $\langle s_2, s_5\rangle$, $\langle s_2, s_6\rangle$, $\langle s_2, s_7\rangle$, $\langle s_2, s_8\rangle$, $\langle s_3, s_5\rangle$, $\langle s_3, s_6\rangle$, $\langle s_4, s_1\rangle$, $\langle s_4, s_6\rangle$, $\langle s_5, s_7\rangle$, $\langle s_6, s_7\rangle$, $\langle s_7, s_1\rangle$, $\langle s_7, s_2\rangle$.
BSP: Is there a module (a collection of compatible sets), such that the total length of the microinstruction would be ≤ 6 bits?
Two design alternatives are considered. The microinstruction that corresponds to the first alternative yields a total length of 7 bits

$$c_1 = \{\langle s_1\rangle\}; \quad c_2 = \{\langle s_4, s_7\rangle, \langle s_2, s_3\rangle\}; \quad c_3 = \{\langle s_5, s_6, s_8\rangle\}.$$

Alternative 2 yields a satisfying solution (with total microinstruction length of 6 bits)

$$c_2 = \{\langle s_2, s_3\rangle\}; \quad c_3 = \{\langle s_4, s_7, s_8\rangle, \langle s_1, s_5, s_6\rangle\}.$$

B. The Intractability of the BSP

The technical result in this section concerns the computational complexity of the BSP. Over the past decade, complexity theory has emerged from a branch of computer science almost unknown to the operations research community into a topic of widespread interest and research. Computational complexity theory seeks to classify problems in terms of the mathematical order of the computational resources—such as computation time, space and hardware size—required to solve problems via digital algorithms. The notion of "easy to verify" but not necessarily "easy to solve" decision problems is at the heart of the Class NP. Specifically, NP includes all those decision problems that could be polynomial-time solved if the right (polynomial-length) "clue" or "guess" were appended to the problem input string. An important subclass of NP problems are referred to as *NP-complete* or Non-deterministic Polynomial time Complete problems [10]. The CPU time required to solve an NP-complete problems, based on known algorithms, grows exponentially with the "size" of the problem. There exists no polynomial time transformations for NP-complete problems, nor are there any polynomial time algorithms capable of solving any NP problems. The potential to solve NP and NP-complete problems depends on the availability of certain heuristics.

In the following, the time complexity of solving the BSP is expressed as a function of the size of the problem. By the size of the BSP we mean the total number of modules and relations $(|M| + |C|)$. We assume that the input length for an instance of a BSP is efficiently encoded (i.e., the problem size grows polynomially with the number of modules and relations). We also assume that each formula $c_i(\langle m_{i1}, m_{i2}, \ldots, m_{in}\rangle_k)$ and E_θ can be verified and computed respectively in polynomial time. It is shown that:

Theorem 1: The BSP is NP-complete.

Proof: Let us prove the first part of the theorem. It is easy to see that the BSP $\in$ NP, since a nondeterministic algorithm need only guess subsets $M_0 \subseteq M$, $C_0 \subseteq C$ and a module $m \in \overline{\langle M_0, C_0\rangle}$ and check in polynomial time that the threshold condition $E_\theta(m) \leq K$ is satisfied.

Next, we transform the *satisfiability problem* (see the appendix) to the BSP. Let $X = \{x_1, x_2, \ldots, x_N\}$ be a set of Boolean variables and $E = \{e_1 \wedge e_2 \wedge \cdots \wedge e_L\}$ be a conjunction of clauses (a clause is defined as the disjunction of literals over X), which form an arbitrary instance of the satisfiability problem. We construct sets M, C, an Evaluation mapping E_θ, and a positive integer K such that there are

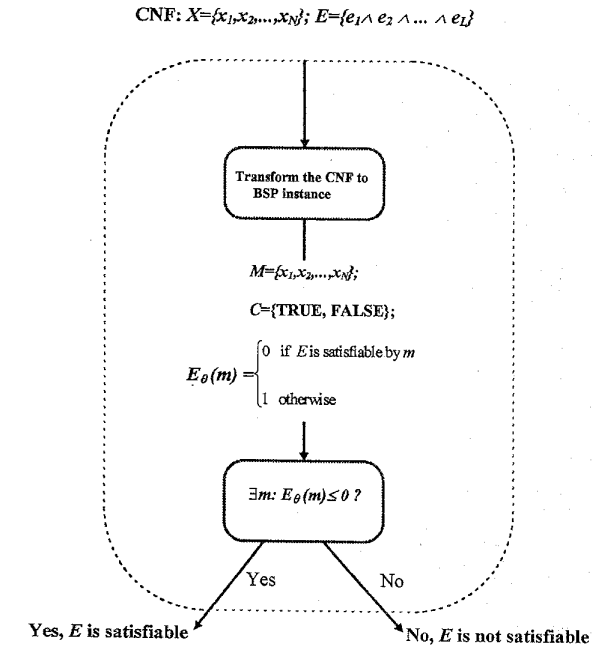


Fig. 14. A Polynomial transformation from SAT to BSP.

Fig. 14. A Polynomial transformation from SAT to BSP.

subsets $M_0 \subseteq M$, $C_0 \subseteq C$ and a module $m \in \overline{\langle M_0, C_0\rangle}$ satisfying $E_\theta(m) \leq K$. The construction goes as follows (depicted in Fig. 14). 1) Let $M = \{x_1, x_2, \ldots, x_N\}$; 2) Let $C = \{\text{TRUE}, \text{FALSE}\}$, such that $\text{TRUE} = \{\langle x_{i_1}, x_{i_2}, \ldots, x_{i_m}\rangle\}$ means $x_{i_j} =$ "true" (hence $\sim x_{i_j} =$ "false") for every $1 \leq j \leq m$ ("FALSE" is defined similarly); 3) a design solution is defined as $m = \{\text{TRUE} = \{\langle x_{i_1}, x_{i_2}, \ldots, x_{i_m}\rangle\}, \text{FALSE} = \{\langle x_{j_1}, x_{j_2}, \ldots, x_{j_{N-m}}\rangle\}\}$ such that $\{x_{i_1}, x_{i_2}, \ldots, x_{i_m}\} \cup \{x_{j_1}, x_{j_2}, \ldots, x_{j_{N-m}}\} = X$ and $\{x_{i_1}, x_{i_2}, \ldots, x_{i_m}\} \cap \{x_{j_1}, x_{j_2}, \ldots, x_{j_{N-m}}\} = \phi$; 4) The evaluation mapping is defined as

$$E_\theta(m) = \begin{cases} 0 & \text{if } E \text{ is satisfiable by } m \\ 1 & \text{otherwise.} \end{cases}$$

Now, one easily verifies that $\exists m : E_\theta(m) \leq 0 \Leftrightarrow E$ is satisfiable. □

Theorem 1 above implies that, in the worst case, the time required to produce a solution is $O(k^n)$ where k is a constant and n, a parameter characterizing the 'size' of the problem. Therefore the use of heuristics (e.g., branch-and-bound techniques or backtrack search) to search for an optimal solution is inevitable when the problem is large.

The intractability of the BSP infers that the number of potential designs is combinatorial–that is, designs are collections of primitive elements, and many different elements can be combined in exponentially different ways:

Proposition 1 (Upper Bound): Let $\Im$ denotes the set of possible solutions (a subset of $\overline{\langle M, C\rangle}$), $c_i \subseteq (M^*)^{n_i}$, $|M| = N$ and $|C| = \rho$. Then $|\Im| \leq \prod_{j=1}^{\rho} 2^{N_i^n}$.

Proof: Every relation c_i consists of assignments, each associated with the Cartesian product of n copies of the set M^*. Since $|(M^*)^{n_i}| = N^{n_i}$, the number of possible assignments is given by $2^{(N^{n_i})}$. As a design solution is the set of ρ relations, the required bound is concluded. □

V. The Constrained Basic Synthesis Problem

A. Problem Formulation

The primitive modules and relations determine the space of possible solutions to the BSP. A designer can only generate structural descriptions that can be formed from these elements. The constraints imposed by the choice of primitive modules and relations are the cause of a fundamental trade-off between the expressiveness of the design representation scheme and the complexity of the BSP. As design representation schemes become more expressive, the space of possible solutions increases.

Here, we consider constraints with respect to three cases: 1) the possible number of assignments that each module can share, grows polynomially (of order γ) with respect to the number of primitive modules N; 2) the cardinality of the relation set C is bounded by some constant ρ; and 3) the cardinality of each relation is bounded by some constant ν. These constraints constitute the constrained basic synthesis problem (CBSP):

Definition 5 (Constrained Basic Synthesis Problem): CBSP is expressed similarly to the BSP by further considering the foregoing constraints.

Formulating the CBSP in this form enables obtaining refined upper bounds of $|\Im|$ (Theorems 2 and 5), and precise conditions where the CBSP can be solved in polynomial time (Proposition 2).

B. Universal Upper Bound on $|\Im|$

In this section, we obtain a *universal* upper bound of $|\Im|$ using an elementary information-theoretic approach. This upper bound is particularly useful when the *a priori* algorithm of solving the CBSP is unknown.

Let us first introduce the important quantity of the entropy of a random variable:

Definition 6: The entropy $H(X)$ of a discrete random variable X drawn according to the probability mass function $p(x), x \in \Omega$, is defined by $H(X) = -\sum_{x\in\Omega} p(x)\log_2 p(x)$. We also write $H(P)$ for the above quantity. The log is to the base of 2 and entropy is expressed in bits.

Theorem 2 (Universal Upper Bound): For $\nu \leq 0.5N^{\gamma+1}$ and sufficiently large N, $|\Im| \leq \nu^{\rho_2{}^{\rho N^{\gamma+1} H_0(\frac{\nu}{N^{\gamma+1}})}} \cdot H_0(P)$ denotes the binary entropy function $H_0(P) = -p\log_2 p - (1-p)\log_2(1-p)$.

Proof: The number of possible assignments constituting any relation $c \in C$ is bounded by $N^{\gamma+1}$. A design solution is defined by a set of ρ relations. Consider each relation c_i contains z_i assignments ($z_i \leq \nu$) out of the $N^{\gamma+1}$ possible ones. Therefore the number of possible designs having this characteristic is given by $\prod_{i=1}^{\rho}\binom{N^{\gamma+1}}{z_i}$. Hence,

$$|\Im| \leq \underbrace{\sum_{z_1=1}^{\nu} \cdots \sum_{z_\rho=1}^{\nu}}_{\rho} \prod_{i=1}^{\rho} \binom{N^{\gamma+1}}{z_i}.$$

It is known that for large values of N, the binomial coefficients satisfy $\binom{N^{\gamma+1}}{z_i} \leq \binom{N^{\gamma+1}}{\nu}$ (recall that $z_i \leq \nu$). It can also be shown [6] that the binomial coefficients satisfy $\binom{N}{x} \leq 2^{NH_0(\frac{x}{N})}$. Therefore, we obtain $\binom{N^{\gamma+1}}{\nu} \leq 2^{N^{\gamma+1}H_0(\frac{\nu}{N^{\gamma+1}})}$. Plugging into the foregoing upper bound on $|\Im|$ and rearranging, we obtain the required inequality. □.

Theorem 2 shows that the cardinality of $\Im$ grows exponentially with $N^{\gamma+1}$ and with the binary entropy $H_0(\frac{\nu}{N^{\gamma+1}})$. The upper bound can be controlled by a suitable selection of N and ν. The binary entropy is a concave function of the distribution, equals 0 when $p = 0$ or 1, and the maximum is obtained when $p = 0.5$. Therefore, the cardinality of $\Im$ can considerably be decreased either by solving problems where ν is small with respect to $N^{\gamma+1}$(i.e., $p = \frac{\nu}{N^{\gamma+1}} \to 0$), or where ν approaches $N^{\gamma+1}$ (i.e., $p = \frac{\nu}{N^{\gamma+1}} \to 1$). Indeed we show that:

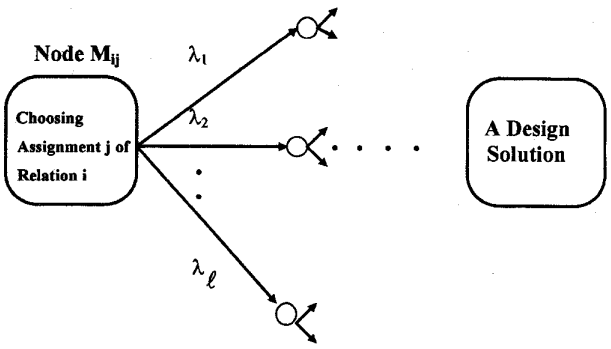


Fig. 15. The probabilistic decision making process.

Proposition 2: For $\nu \ll N^{\gamma+1}$ and sufficiently large N, $|\Im|$ is bounded by: $\alpha\exp(o(N))N^{\gamma+1\nu\rho}$, α is a constant and $o(N)$ means $\frac{o(N)}{N} \underset{N\to\infty}{\longrightarrow} 0$.

Proof: The proof is similar to that of Theorem 1, except for using the following useful approximation when needed: $\binom{N^{\gamma+1}}{\nu} \approx (\exp(\nu)/\sqrt{2\pi}\nu^{(\nu+0.5)})\exp(o(N))N^{(\gamma+1)\nu}$. Notice that the first term is a constant. □

VI. Refined Upper Bound on $|\Im|$

A. Probabilistic Design Selection

In this section, we tighten the universal upper bound in the last section by taking into account that the designer uses probabilistic heuristic strategy to search for solutions for the CBSP.

The nature of the information involved in the search for a design solution may be *deterministic*, by showing which designs in $\Im$ are categorically inferior, or *probabilistic*, by identifying those designs having the greatest probability of solving the problem. The probabilistic decision making process is supported by many protocol studies on design [e.g., 24]. In a probabilistic framework, the designer finds himself at a certain decision stage, labeled Node $\mathbf{M}_{ij}$ in Fig. 15. Node $\mathbf{M}_{ij}$ is associated with a set of assignments (constituting relation c_i), λ_1 through λ_ℓ, each with a probability p_i of being included in a successful design solution. The choices are made out of the $N^{\gamma+1}+1$ possibilities (including the void assignment). We assume that the process of decision making is sequential, and that the decisions are mutually independent. Thus, choosing $\nu\rho$ assignments results in a design solution. The designer's problem is that he may not know which assignments will ultimately lead to a satisfying design solution. In other words, he has little or no information on the value of the p_i's.

In the sequel, we establish the appealing proposition that not all possible solutions in $\Im$ have the same probability of solving the CBSP and that the probability of identifying a solution for the CBSP is inversely proportional to the number of assignments specified.

B. The Asymptotic Equipartition Property (AEP)

Let us first recall some basic results of information theory deriving from the Asymptotic Equipartition Property (AEP) which is formalized as follows:

Theorem 3: If $X_1, X_2, \ldots$ are i.i.d.$\sim$ $p(x), x \in \Omega$, then $-\frac{1}{n}\log p(X_1, X_2, \ldots, X_n) \to H(X)$ in probability.

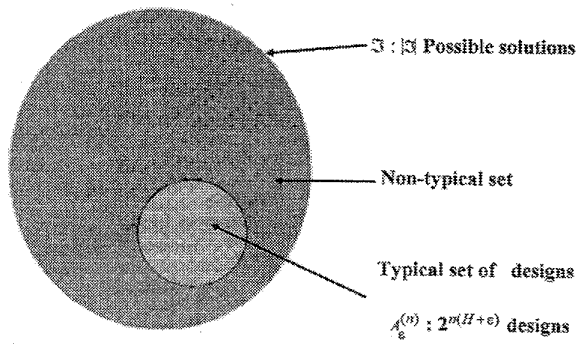


Fig. 16. Typical set of design solutions.

Proof: By the weak law of large numbers. □

Definition 7: The typical set $A_\varepsilon^{(n)}$ with respect to $p(x)$ is the set of sequences $(x_1, x_2, \ldots, x_n) \in \Omega^n$ with the following property:

$$2^{-n(H)(X)+\varepsilon)} \leq p(x_1, x_2, \ldots, x_n) \leq 2^{-n(H(X)-\varepsilon)}.$$

As a consequence of the AEP, it can be shown that the set $A_\varepsilon^{(n)}$ has the following properties:

Theorem 4:

1) If $(x_1, x_2, \ldots, x_n) \in A_\varepsilon^{(n)}$, then $H(X) - \varepsilon \leq -\frac{1}{n}\log p(X_1, X_2, \ldots, X_n) < H(X) + \varepsilon$.
2) $\Pr\{A_\varepsilon^{(n)}\} > 1 - \varepsilon$ for n sufficiently large.
3) $|A_\varepsilon^{(n)}| \leq 2^{n(H(X)+\varepsilon)}$, where $|A|$ denotes the number of elements in the set A.
4) $|A_\varepsilon^{(n)}| \geq (1-\varepsilon)2^{n(H(X)+\varepsilon)}$ for n sufficiently large.

Proof: The proof can be found in [6, pp. 52–53], and is omitted here. □

Thus the typical set has probability nearly 1, all elements of the typical set are nearly equiprobable, and the number of elements in the typical set is nearly $2^{nH(X)}$. Moreover, any property that is proved for the typical sequences will then be true with high probability and will determine the average behavior of a large sample.

C. Consequences of the AEP on the CBSP

Let us now apply the AEP to the CBSP considering a probabilistic decision making process as explicated above. We let $n = \nu\rho$ and $A_\varepsilon^{(\nu\rho)}$ denotes the typical set. The typical set is interpreted as the *smallest* set of design solutions having the *highest* probability of including a solution for the CBSP (see Fig. 16). Assuming that the selections of assignments are independently drawn from the probability mass function $p(x)$ and that $\nu\rho$ is sufficiently large, we obtain $|A_\varepsilon^{(\nu\rho)}| \approx 2^{\nu\rho H(X)}$.

In case the selections are uniformly distributed over the set of possible assignments, (i.e., $p(x) = 1/(N^{\gamma+1}+1)$), we obtain $|A_\varepsilon^{(\nu\rho)}| = 2^{\nu\rho\log(N^{\gamma+1}+1)} \approx N^{(\gamma+1)\nu\rho}$. Thus, we see that while the universal upper bound on the size of $\Im$ grows exponentially with respect to $N^{\gamma+1}$ (Theorem 2), the typical set of designs is a fairly *small* set that has the highest probability of including a solution for the CBSP. It is interesting to note that the cardinality of the typical set is similar to the upper bound derived in Proposition 2 (where $\nu \ll N^{\gamma+1}$).

VII. Summary

This work develops a model for the design process. Such a model enables formal understanding of design processes, and as such enables modification of the design to various needs. For example, modifying the input specifications to consider robotics assembly can result in a different design solution (see [19]). The paper develops a model of the process based on triple interleaved activities of analysis, synthesis and evaluation, which explode the specification world, and accordingly the design process, until a solution is achieved. The basic synthesis problem (BSP) is defined and shown to be generally intractable. Expressions for the cardinality of the set of possible solutions are obtained, under various conditions. Our results are essential for developing heuristic strategies to search for optimal and suboptimal design solutions. Our main conclusion is that although expressiveness of the BSP is necessary to allow for the generation of a wide variety of designs, simply increasing the expressiveness of a design problem swamps the designer with alternatives. So, any increase in expressiveness must be accompanied by an increase in the designer's ability to control the complexity of the design space.

Appendix
(The Satisfiability Problem)

The satisfiability (SAT in short) is expressed in terms of the following:

- A set $X = (x_1, x_2, \ldots, x_N)$ of *Boolean variables.*
- *Literal*—A variable x or its negation $\sim x$.
- *Clause* over X—Defined as the disjunction of literals over X, denoted by e.
- *Expression*—Defined as a conjunction of clauses, denoted by a collection of clauses, $E = (e_1, e_2, \ldots, e_M)$.
- *Satisfying truth assignment* for E—A collection E of clauses over X is *satisfiable* iff there exists true assignment for X that simultaneously satisfies all the clauses in E.

Now, SAT can be formulated as: Given a set X of variables and a collection E of clauses over X. Is there a satisfying truth assignment for E.

References

[1] A. Apple, "Modeling in three dimensions," *IBM Systems Journal*, vol. 7, nos. 3–4, pp. 310–321, 1968.
[2] M. Asimow, *Introduction to Design.* Englewood Cliffs, NJ: Prentice-Hall, 1962.
[3] B. Bohem, *Software Engineering Economics.* Englewood Cliffs, NJ: Prentice-Hall, 1981.
[4] D. Braha and O. Maimon, "A mathematical theory of design: Modeling the design process (Part II)," *Int. J. General Syst.*, 1995.
[5] I. C. Braid, "The synthesis of solids bounded by many faces," *Comm. ACM*, vol. 18, no. 4, 1975.
[6] T. M. Cover and J. A. Thomas, *Elements of Information Theory.* New York: Wiley, 1991.
[7] R. D. Coyne, M. A. Rosenman, A. D. Radford, M. Balachandran and J. S. Gero, *Knowledge-Based Design Systems.* Reading, MA: Addison-Wesley, 1990.
[8] S. R. Das, D. K. Banerji and A. K. Chattopadhyay, "On control memory minimization in microprogrammed digital computers," *IEEE Trans. Comput.*, vol. C-22, no. 9, pp. 845–848, 1973.
[9] C. L. Dym and R. E. Levitt, *Knowledge-Based Systems in Engineering Design.* New York: McGraw-Hill, 1993.
[10] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness.* San Francisco, CA: W. H. Freeman, 1979.
[11] J. H. Garret and M. L. Mehar, "Knowledge-based systems in design and planning," *J. Comp. Civ. Engrg.*, vol. 5, pp. 1–3, 1991.
[12] A. Grasselli and U. Montanari, "On the minimization of read-only memories in microprogramed digital computers," *IEEE Trans. Comput.*, vol. C-19, no. 11, pp. 1111–1114, 1970.
[13] J. P. Hayes, *Computer Architecture and Organization.* New York: McGraw-Hill, 1988.
[14] J. C. Jones, *Design Methods: Seeds of Human Futures*, 2d ed. New York: Wiley, 1980.
[15] C. S. Krishnamoorthy, H. Shivakumar, S. Rajeev and S. Suresh, "A knowledge-based systems with generic tools for structural engineering," *Structural Eng. Rev.*, vol. 5, no. 1, pp. 334–358, 1993.

[16] M. A. Lavin and L. I. Lieberman, "A system of modeling three dimensional objects," IBM Research Report RC-5765, 1975.
[17] O. Maimon and D. Braha, "A proof of the complexity of design," *Kybernetes*, vol. 21, pp. 59–63, 1992.
[18] ——, "A mathematical theory of design: Representation of design knowledge (Part I)," *Int. J. General Syst.*, 1995.
[19] ——, "An exploration of the design process," Tech. Rep. 94-3, Dept. Manuf. Eng., Boston University, Boston, MA, 1994.
[20] W. H. Middendorf, *Design of Devices and Systems*. New York: Marcel Dekker, 1986.
[21] C. V. Ramamoorthy *et al.*, "Issues in the development of large, distributed and reliable software," in *Advances in Computers*, vol. 26, pp. 393–443, M. C. Yovits, ed. New York: Academic Press, 1987.
[22] H. A. Simon, *The Science of the Artificial*. Cambridge, MA: MIT Press, 1981.
[23] D. Sriram and K. Cheong, "Engineering design cycle: A case study and implications for CAE," in *Knowledge Aided Design*. New York: Academic, 1990.
[24] D. Ullman, T. G. Dietterich and L. A. Stauffer, "A model of the mechanical design process based on empirical data," *AI EDAM*, vol. 2, pp. 33–52, 1988.
[25] R. K. Vemuri, Soo-Ik Oh and R. A. Miller, "Topology-based geometry representation to support geometric reasoning," *IEEE Trans. Syst., Man, Cybern.*, vol. 19, no. 2, pp. 175–187, 1989.

Abstractions of Finite-State Machines and Optimality with Respect to Immediately-Detectable Next-State Faults

Kostas N. Oikonomou

***Abstract*— Abstraction is the process of lumping together some of the inputs, states, and outputs of a finite-state system (machine) M to transform it into a smaller, generally nondeterministic system M_A. Theoretically, abstraction can be viewed as a method for approximate system simplification, and practically it finds application to system monitoring. Large and complex systems are usually observable through restricted interfaces, which allow an observer only a lumped (abstracted) view of the system, and render some erroneous behaviors undetectable. In spite of the nondeterminism introduced by the abstraction, there is still a class of faults in the system (changes in the next-state or output maps) which are *immediately-detectable* upon occurrence. Here we study the problem of finding an abstraction for an fsm which groups the inputs, states, and outputs into a specified number of classes, while maximizing the number of immediately-detectable (i.d.) next-state faults of multiplicity 1. Assuming that a partition of the machine's outputs is given, we show that the optimal choice of either the state or the input partition is an *NP*-hard or *NP*-complete problem. However, we give a polynomial-time algorithm that finds an approximately optimal partition of the machine's inputs for any given partition of the states. We also provide a bound on the optimum, computable in polynomial time. Numerical experiments with the algorithm on randomly-generated machines with two types of state partitions, suggest that (a) the optimal number of i.d. next-state faults increases linearly with the number of blocks of the input partition, and (b) that more faults are i.d. in machines with "sparse" structure and with less uniform state partitions.**

I. Introduction

Finite-state machines are useful as models of many real systems. An *abstraction* of a finite-state machine (fsm) M consists in lumping (aggregating) some of its states, inputs, and outputs into classes, which then become the states, inputs, and outputs of a smaller fsm M_A. If M is deterministic, the abstracted machine M_A will, in general, be nondeterministic. If the lumping is carried out in accordance with appropriate criteria, abstraction can be viewed as a general method for *approximate simplification* of a large and complex finite-state system (modeled by the fsm).

An alternative viewpoint is that the concept of an abstraction represents the situation in which an observer is monitoring a complex system: generally the observations possible on the system are *limited* (not all details are visible), and the observer's knowledge of the system's correct behavior is *incomplete*. We will see that the abstraction can represent the interface limitations, as well as the observer's simplified model of the system.

The first viewpoint, abstraction as approximate simplification, is of theoretical interest, while the second is more practical, being applicable, for example, to the monitoring of systems for the purposes of fault detection. The concept of abstraction was introduced originally in this context in [15][1]. Whichever of these two viewpoints is adopted, the purpose of this paper is to present some criteria for selecting an *optimal* abstraction of a given fsm, to show that the choice of an optimal abstraction is computationally hard, and to present an algorithm which computes an approximately optimal abstraction in reasonable time.

A. Abstractions, Observers, and Optimality

Given a system of interest, let the fsm M represent the system in full detail. (Of course, every model of a real system is itself at some level of abstraction.) The abstracted fsm M_A representing the system at a higher level of abstraction is constructed by lumping M's states, inputs, and outputs into classes. If Q, X, Y are the sets of states, inputs, and outputs of M, these classes are specified by *partitions* Π_Q, Π_X, and Π_Y. We call the triple (Π_Q, Π_X, Π_Y) the *abstraction* A. The machine M_A is defined as follows. Let Q_i and X_j be blocks of Π_Q and Π_X respectively. To see if M_A can move from state Q_i to state Q_k with input X_j, check if M has a transition from some state in Q_i to some state in Q_k with some input in X_j. M_A's output map is defined similarly. Clearly, the abstraction generally results in a nondeterministic machine. Otherwise, the fsm M and abstraction A are completely arbitrary. If $M = (Q, X, Y, \delta, \lambda)$, with δ and λ the next-state and output maps, we will denote M_A by $(\Pi_Q, \Pi_X, \Pi_Y, \delta_A, \lambda_A)$.

Example: Fig. 1 shows a 4-state machine, behaving like an up/down mod 3 counter. An abstraction A is shown that lumps the states into two classes and the outputs into three classes, while leaving the inputs as they are. The machine M_A and its next-state and output maps δ_A and λ_A are also shown.

Now imagine a human or machine entity observing the system M; naturally, we will refer to it as the "observer". M is subject to malfunction, and the observer's task is to detect erroneous behaviors in M. However, the observer only sees the states, inputs, and outputs of M_A, and we will assume that M_A is all that it knows about the *specification* of the system. Thus the observer's knowledge of the system's correct operation is commensurate with its observational abilities. The situation is shown in Fig. 2. Note that the observer's limitations result either from the fact that the interface to M is restricted, or because of a desire to have a simplified model of M. We will further assume that the observer observes only *single transitions*

Manuscript received August 1, 1993; revised April 24, 1994, and January 19, 1995.
The author is with AT&T Bell Laboratories, Holmdel, NJ 07733 USA.
Publisher Item Identifier S 1083-4427(96)00064-1.

[1] For the monitoring approach to fault detection in computing systems see [1], [2], [5], [12], [20].