Research in Engineering Design (1995) 7: 102–125

Research in
Engineering
Design

Recursive Annealing: A Computational Model for Machine Design

Linda C. Schmidt and Jonathan Cagan*
Department of Mechanical Engineering, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA

Abstract. *We propose a model for optimally directed conceptual design of machines in which the transformation of function to form occurs iteratively along an abstraction continuum. An algorithm called FFREADA is introduced as a computational implementation of the model. FFREADA is a grammar-based optimizing design algorithm that uses recursive simulated annealing to generate optimally directed designs. During FFREADA's design process, the mapping of function to form is accomplished using an abstraction grammar production system and a predefined library of function and form entities. FFREADA also has a random design generation mode that can be used to record data to characterize the space of design solutions. FFREADA is demonstrated by designing an idealized power supply using a variety of performance objectives. Results show the algorithm able to explore and record information about a tractably infinite design space before converging to the optimal design.*

Keywords. Abstraction; Conceptual design; Design theory; Grammars; Simulated annealing

1. Introduction

The conceptual design process is an integral part of the mechanical engineering discipline and yet, it remains an activity with little consensus on effective paradigms for the development of computational design tools. One compelling area of conceptual design research is machine design, the process of transforming the description of desired machine function into a description of a machine. A machine is a mechanical system that performs a specific task, transforming and transferring motion and force.

We propose a computational model for optimally directed[1] conceptual design of machines in which the transformation of function to form occurs iteratively along an abstraction continuum (Gero *et al.*, 1991; Ullman 1992) on a number of defined levels of abstraction. The levels of abstraction are arranged from highest and most abstract, to lowest and least abstract. Thus ordered, these levels form a hierarchy of design arenas. This design model requires:

1. a representation scheme for machines at varying levels of abstraction;
2. a methodology for generating hierarchically consistent designs at each level of abstraction; and
3. a means of propagating evaluation feedback from the form level of abstraction to higher levels.

FFREADA (Function to Form REcursive Annealing Design Algorithm) is a recursive implementation of this model of design. With FFREADA, machine components are represented as form entities and are assigned to the lowest level of design abstraction. The form entities are systematically abstracted into increasingly general function entities and assigned to one of the levels of abstraction. An *abstraction grammar*, formalized as a string grammar, is used to generate feasible machine designs at each level of abstraction from the applicable entities. Design generation occurs from the top level of the abstraction hierarchy to the form level, with each design generation using the prior level design as a pattern. A set of designs is created, one on each level of abstraction. Designs on levels of abstraction above the form level are primarily functional descriptions of machine activity and are called function structures (used in this work as defined by Pahl and Beitz, 1988). Designs on the form level of abstraction are arrangements of form entities representing machine components and can be evaluated directly using an objective function.

Simulated annealing, a stochastic optimization technique, is used during the form level design process to find the best form instantiation[2] of the last pattern

*Correspondence and offprint requests to: J. Cagan, Department of Mechanical Engineering, Carnegie Mellon University, Pittsburgh, PA 15213, USA.

[1] "Optimally directed" is defined as an approach to design that attempts to focus the search for design solutions on the area of the design state space in which the optimal solution lies. A search conducted in this way will find a *near-optimal* solution without necessarily finding *the* optimal solution (Cagan and Agogino, 1991).

[2] Instantiate means to represent an abstraction by a concrete instance. An instantiation is one such instance.

design. This requires an automatic design generation mechanism so that large numbers of designs are generated. The abstraction grammar is the means by which feasible designs can be generated. The simulated annealing algorithm controls the design process. Simulated annealing is also applied between the levels of abstraction. Design evaluation feedback from the form level is propagated to higher levels of abstraction to select pattern designs that ultimately converge to good form designs. At any time during the algorithm there are multiple design processes occurring, each being controlled by an annealing process. We call this *recursive annealing.*

2. Background

Design theory literature is replete with design models and methodologies. Mechanical engineering design theory research is divided into six categories by Finger and Dixon (1989a, 1989b). The three categories applicable to design processes are:

1. descriptive models of the design process (e.g., protocol studies, cognitive models, systematic design and case studies);
2. prescriptive models of the design process (e.g., canonical design, morphological analysis and axiomatic design); and
3. computer-based models of the design process.

Building design models based on what is done (descriptive), what ought to be done (prescriptive) or what can be implemented via computer (computer-based) can be short-sighted. We assert that the best models are those which incorporate aspects of all three approaches, giving the models the advantages of each approach. Such integrated models should be the focus of design theory research.

2.1. Recent Work on Transformed Design Models

Within the broad category of design process models, we observe many different paradigms for mechanical engineering design. Common ones include design from first principles, design from prototypes, design based upon qualitative reasoning, grammatical design and, more recently, models based on the use of abstraction. Machine design is by its nature transformational; these models all perform some type of transformation from initial specifications of machine function to an arrangement of machine components.

Transformational models of design provide structured methods for mapping function to form, often simplifying the problem by decomposing it into a hierarchical one. Our structured hierarchical abstraction approach finds its origins in the systematic design method of Pahl and Beitz (1988), whose representational concept of energy, material and signal flows serves as the basis for the entity representation presented in Section 4. Hundal (1990) and Hundal and Langholtz (1992) attempt to implement the initial stages of the systematic method by computer, adding a degree of automation. Hoover and Rinderle (1989) present a synthesis strategy for design that bases transformations on functional descriptions of components and also represents incidental component behavior. This strategy provides the means for attempting to achieve a level of functional integration or function sharing. Iyengar *et al.* (1994) and Bradley *et al.* (1993) also propose hierarchical design models. Iyengar *et al.* focus on recombining aspects of top-down, functionally based representations of a design so that alternatives, transformed from the same functional network, can be evaluated and compared. The Schemebuilder software of Bradley *et al.* is a mechatronics designer's aid package that uses a hierarchical functional approach to model a system and suggest a range of functional solutions.

A new paradigm, grammatical design, grew from the use of grammars in architecture to describe languages of design and is appearing in a variety of approaches to engineering design. Finger and Rinderle (1989) propose a bond graph grammar, a graph-based language, for mechanical design and apply it to gear box design. They have found the formal nature of a grammatical approach appealing. Mullins and Rinderle (1991) pursue the use of grammars again for the benefits of their formality and for their inherent transformational nature. Longenecker and Fitzhorn (1991) use a shape grammar to define a solid model representation to generate only realizable shapes. In this case, a grammar's ability to define a specific language of designs is exploited. In a more specific application, Brown *et al.* (1993) create a labeled parametric shape grammar to represent objects manufacturable by a lathe. Andersson (1993) takes a broad grammatical approach and creates the vocabulary for a general conceptual design grammar. The vocabulary is the basis for a language of design that is not limited to geometric descriptions. Cagan and Mitchell (1993) combine a shape grammar with simulated annealing to create shape annealing, a way to generate an optimal shape. This work was built on by Reddy and Cagan (1994) where shape annealing was used in an engineering application – the generation of optimal trusses.

Abstraction is recognized as a powerful tool in

design processes in general, and in mechanical design in particular (Paz-Soldan and Rinderle 1989; Hoover *et al.* 1991). During machine conceptual design, designers work at a high level of abstraction, adding detail and applying constraints to map functional specifications into forms. Responding to the importance of abstraction to the design process, abstraction-based design can be considered a newer paradigm in the mechanical engineering field. Schmidt and Cagan (1992) and Snavely and Papalambros (1993) use abstraction as a basis for the transformation of functional specifications into form in the conceptual design process and the configuration design process, respectively. Abstraction is recognized as a tool to characterize the complex relationship between form and function which lies at the heart of the transformational methods (see Rinderle 1986; Ulrich and Seering 1988; and Flemming *et al.* 1992).

2.2. Desirable Properties of a Design Model

A design model should incorporate the best of what designers currently do, what they should do for best results and what they can do given present technology. Incorporating the use of abstraction in a design model imitates present design practice. Addressing cognitive issues and computing issues within the same design model is also possible and desirable.

Consider cognitive issues in design; most proposed design models and their implementations fail to encourage practices known to increase the potential for good design solutions, such as lateral thinking. Successful conceptual design relies on the ability to consider conceptually diverse design solutions. A common phenomenon preventing solution diversity as identified by Adams (1986) is the tendency to delimit a problem area too closely, applying perceived constraints to the problem. The type of thinking that enables a widening of consideration is termed lateral thinking. The need for systematized lateral thinking is heightened by a second common design phenomenon called *design fixation*, the adherence to a limited set of ideas during the design process (Jansson and Smith 1991). Jansson and Smith have empirically verified the existence of design fixation in design students as well as professional design engineers and suggest that design methodologies be adapted to assist designers in overcoming this problem.

In the past, design models could be unambitious in the search for a best design solution, claiming instead the practicality of finding a satisfactory solution quickly. With few exceptions (Cagan and Agogino 1987, and Schmidt and Cagan 1992, among them) the design paradigms most common in the literature provide for the deterministic generation of one feasible design solution from a knowledge-intensive or decision-intensive process. Given that the ultimate goal of the design process is to arrive at the *best* solution possible with current resources, it is natural to build optimization practices into modern design process models.

3. A Recursive View of Conceptual Design

We propose that machine design occurs iteratively on discrete levels of abstraction along the *abstraction continuum* and that each design exists as the instantiation of a design created at a higher level of abstraction (Fig. 1). The abstraction continuum orders designers' images of a machine according to the level of explicit functional detail they embody. At the

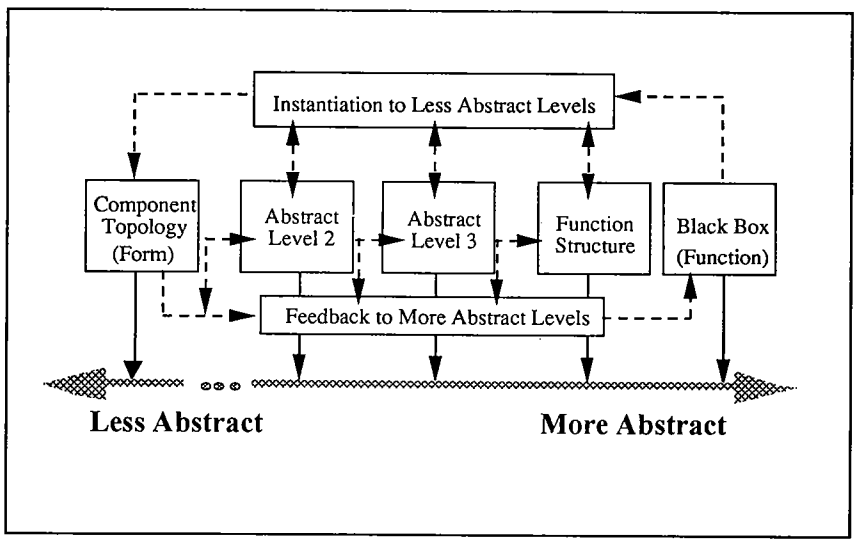

Fig. 1. Conceptual design along the abstraction continuum.

highest level of functional abstraction, a machine is a black box image of itself; inputs and desired outputs are known, as is a general functional description of the transformations occurring inside the black box. On the lowest level of functional abstraction, the image is a component assembly description, not unlike a machine blueprint. This design image accurately details the form of the machine and yet holds no explicit functional description. Ordered between these two extremes, from the black box to the component layout description, are machine representations of increasingly detailed functional description until the detail becomes so explicit that it is instantiated into a component form providing the expressed functionality. In this view, each design image exists as a feasible, yet not necessarily unique, instantiation of the designs at higher levels of abstraction. This process of instantiation along the abstraction continuum preserves design consistency, the characteristic that high-level functional decisions are satisfied by instantiated component descriptions.

Direct decomposition into functional descriptions (function structures) and subsequent instantiation into component forms in a single traversal along the abstraction continuum is prohibited by the complexities inherent in mechanical design. Instead, we observe design iterations along the abstraction continuum. These iterations are motivated by both the lack of articulable performance metrics at high levels of abstraction and the lack of uniqueness in transformation of more to less abstract functional structures. Specifically:

1. There are no clear performance metrics for abstract designs. On the form level of abstraction, evaluation of a candidate design is straightforward; machine performance metrics include, but are not limited to: size, weight, power requirements, efficiency, capacity for force generation and economic features. Unfortunately, precise performance metrics are difficult to articulate for designs that are completely described as function structures.

2. There is a many-to-many mapping from function to form along the continuum. This multiplicity is compounded by *function coupling* or *sharing* in which physical components satisfy multiple functions simulaneously (Ulrich and Seering 1988; Rinderle 1986).

In practice, designers discretize the abstraction continuum and work on a finite number of abstraction levels, each level having a set of function or machine component representations embodying descriptive characteristics appropriate to that level of abstraction. The ensuing iterative design process begins at a highly abstract level (e.g., a function structure level) with instantiation from one abstract level to the next lower level along the continuum, until the design is instantiated on the form level. Results of evaluation of the form design are noted and linked to functional decisions made at higher levels of abstraction. The designer then returns to more abstract design images, modifies the functional design and explores the ramifications of the change by repeating the instantiation process until the new design is also evaluated. In this way, abstract design alternatives are explored more thoroughly. As the designer gains insight, better design decisions are made and the design converges to a final machine component layout.

4. Abstraction Grammar

We define an abstraction grammar as a production system for the representation and generation of function and form layouts. Like other grammars, the abstraction grammar has a vocabulary of valid symbols and rules that govern the arranging of those symbols into meaningful expressions. An abstraction grammar is defined on a predetermined and ordered set of abstraction levels, each level characterized by the type of functional information that its assigned symbols contain. The symbols of each abstraction level describe machine component characteristics embodied on that level of abstraction, emphasizing certain features according to a stated representation scheme. A design at one level of abstraction is generated by instantiating a design from the next higher level and, as such, is typically one of many possible instantiations. The instantiated design serves as a pattern for design on the next lower level of abstraction.

A set of machine components forms the basis for the grammar's library of vocabulary symbols. The machine components are represented as form entities which exist only on the lowest level of abstraction (numbered level 0). A form entity is encoded with one component's relevant functional characteristics, individual identification information and knowledge sufficient for performance evaluation of a machine in which this component operates. Form entities are then abstracted, omitting some detail and eliminating subsequent duplication, so that a set of entities is created for the next higher level of abstraction in the hierarchy of levels. This abstraction process focuses on the functions performed by the machine components. The entities on levels above the form level are called function entities. In addition to creating the entity sets in this bottom-up fashion, a designer may

use a top-down approach, adding functional entities of interest at any level of abstraction. One can then observe how the existing set of form entities can satisfy the new variants.

It is possible to construct a grammar that manipulates the form level entities into machines, bypassing the functional patterns provided by the higher levels of abstraction in this design model. However, the hierarchical design record provided by designing on each successive level of abstraction allows the designer to monitor the effectiveness of individual functional entities and patterns. This design approach begins to address the fundamental problem of articulating performance metrics for function structures by imposing a clear association between functions and the forms by which instantiation is achieved.

4.1. Function and Form Representation

Essential to the abstraction grammar is the representation scheme used on each level of abstraction. Design theory literature is abundant with proposed function and form representations. Their diversity is illustrated here by citing a few examples. Kannapan and Marshek (1990) propose an algebraic and predicate logic representation scheme for machine elements with design synthesis occurring by satisfaction of prescribed machine element relationships. Pahl and Beitz's (1988) systematic design approach describes machines as arrangements of functions, called function structures, each acting upon the energy, material and signal flows through the machine. Component assemblies, termed solution concepts, are cataloged and accessed according to the subfunctions each satisfies. Behavior graphs (Welch and Dixon 1992) combine behavioral reasoning based on qualitative physics with the power of bond graphs to represent physical behaviors. A qualitative representation of machine behaviors (Kurumatani *et al.*, 1990) uses qualitative reasoning to represent behavior as a series of states. In the same work, geometry is represented using two types of graphs, a geometric transition graph and a geometric connection graph.

Bond graphs (Paynter 1961), first developed for analysis of multiport dynamical systems, are, not surprisingly, often used to represent mechanical systems incorporating electrical, hydraulic, pneumatic and magnetic components. But in addition to representing these multi-domain systems, modeling applications have been extended to bioengineering, chemical, heat transfer, thermodynamic, fluid mechanics, solar and nuclear systems (Breedveld *et al.* 1991 and Montbrun-Di Filippo *et al.* 1991). The ubiquity of bond graph modeling suggests that the representation has desirable capabilities, two of which are the applicability to devices and components of many different domains, and the ability to change granularity (e.g., bond graphs can represent an entire passenger car as in Rosenberg and Zalewski 1986, or its automatic transmission power train, its manual transmission gearbox, or the transmission clutch's electrohydraulics as in Hrovat and Tobler, 1991). Representations that seek the same broad range of modeling capability at various levels of abstraction must incorporate these characteristics.

Representation systems are commonly introduced in conjunction with new design process models; the success of the latter is often dependent on the adequacy of the former. The success of this work is not dependent on the introduction of a new function and form representation system for use with FFREADA. Rather, we believe that existing representation systems can be articulated as part of a design grammar.

The abstraction grammar representation system requires a library of both function and form entities. Function entities are abstract images of machine components, highlighting one or more functions performed by the component. For example, a rack and pinion's function can be described as converting reciprocating translational motion into reciprocating rotational motion. By excluding some of the functional detail, it also belongs to the class of entities that convert translational motion to rotational motion. Some functional detail is abstracted out of the machine component to create entities such as "anything that converts translational motion to rotational motion". The grammar rules combine the entities of each level of abstraction into meaningful structures that describe machine functioning in terms characteristic of that abstraction level. The resulting structure is a design of a machine on one particular level of abstraction. It exists as an instantiation of the design of the same machine on the next higher level and serves as a pattern for designs on the next lower level.

To illustrate these ideas further, consider adapting the Pahl and Beitz (1988) energy, material and signal flow representation system for generation of designs at different levels of functional abstraction. Parts (a) and (b) of Fig. 2. are recreations of Pahl and Beitz's function structures for a tensile testing machine. They serve as an example of establishing physical flows of energy (E_{load} and $E_{\text{deformation}}$), material (Specimen) and signal (S, S_{force} and $S_{\text{deformation}}$) to define the functional representation of a machine. Under the proposed entity representation system of the abstraction grammar, Pahl and Beitz's Fig. 2(a) becomes the statement of specifications of the design problem and Fig. 2(b) is a function structure at the highest level of

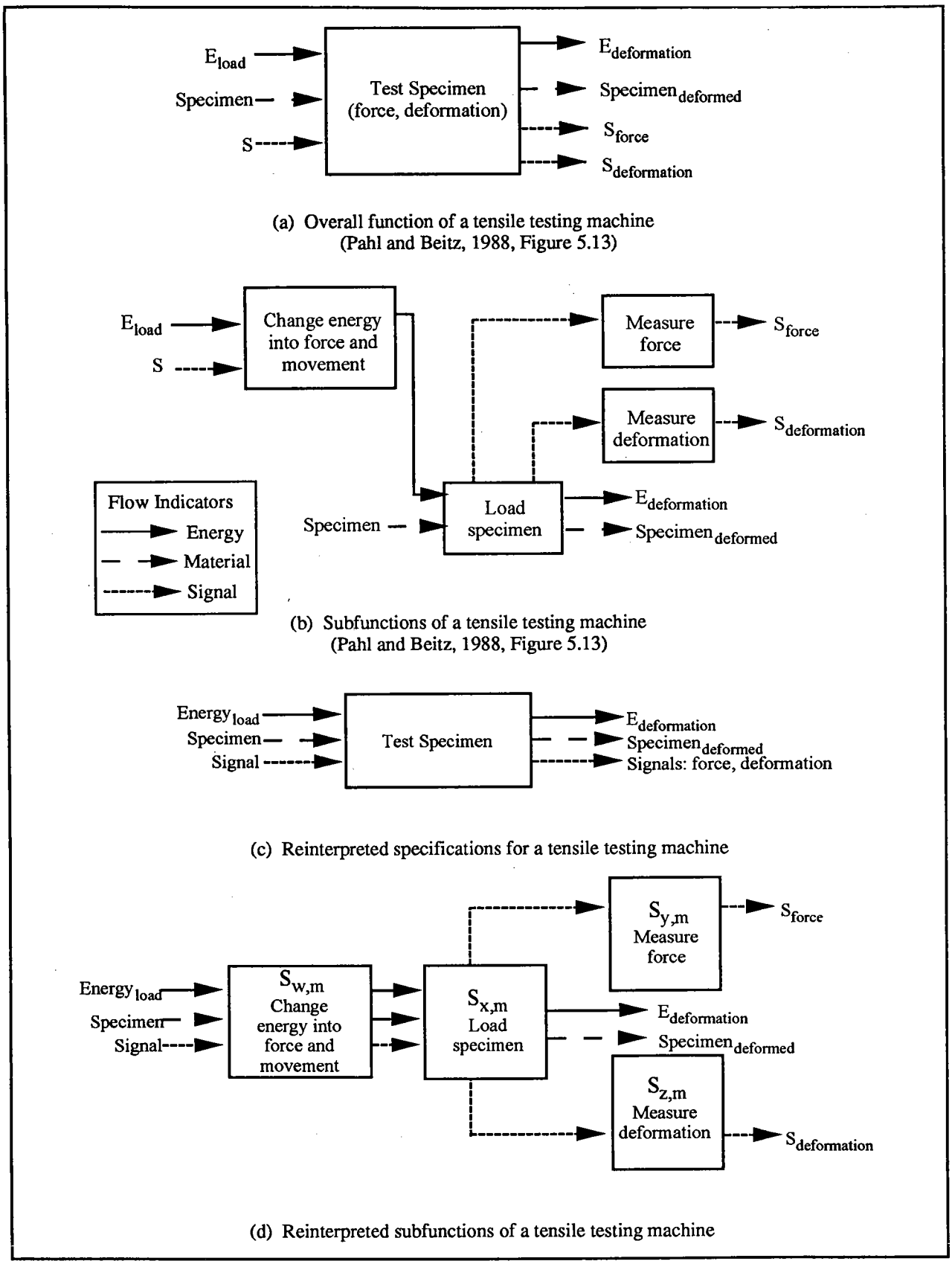

Fig. 2. Pahl and Beitz function structures for a tensile testing machine reinterpreted.

abstraction which we might define as the flow level. These reinterpretations of the Pahl and Beitz figures are shown in Fig. 2 parts (c) and (d).

The tensile testing machine of Fig. 2(d) could be described by a variety of spatial grammars. The layout which features a parallel arrangement of function structure entities might lend itself to a graph grammar representation. If one allowed energy, material and signal flows to pass through some entities unchanged, we could align the function structure into a serial arrangement of components and represent that arrangement by the string $S_{w,m}S_{x,m}S_{y,m}S_{z,m}$, where m is the level of abstraction of the entities in the string. Many machines of interest exist as a serial arrangement of components, especially machines designed according to a process sequence. Although the concepts of the representation are not limited to serial arrangements of entities, we have chosen to limit the initial exploration of this work to *serial machines* and, in doing so, choose to formalize our abstraction grammar as a string grammar.

The strings in the grammar are a serial arrangement of entity symbols, each symbol representing a collection of sets of information parameters. Let $S_{i,j}$ be the symbol for the "ith" entity in the library on level of abstraction "j". There are $n+1$ levels of abstraction in the grammar, levels $0, 1, \ldots, n$.

$$S_{i,j} = A_{i,j} \cup P_{i,j} \cup K \cup M,$$

where

$$A_{i,j} = \{a_j, a_{(j+1)}, \ldots, a_n\} \cup \{b_j, b_{(j+1)}, \ldots, b_n\} \cup \{c_j, c_{(j+1)}, \ldots, c_n\},$$

a set of activation parameters describing the energy (a_j's), material (b_j's), and signal (c_j's) flows input to the entity, with n being the top level of abstraction in the grammar. Each of the a_x, b_x and c_x parameters describe some characteristic of energy, material, or signal flow on the xth level of abstraction. The set $A_{i,j}$ is structured such that

$$A_{i,(j-1)} = \{A_{i,j}\} \cup \{a_{j-1}, b_{j-1}, c_{j-1}\} \in S_{i,(j-1)},$$

for $j = 2, 3, \ldots, n$. $A_{i,0} = A_{i,1}$ because there are no new activation parameters unique to the form level of abstraction.

$$P_{i,j} = \{p_j, p_{(j+1)}, \ldots, p_n\} \cup \{q_j, q_{(j+1)}, \ldots, q_n\} \cup \{r_j, r_{(j+1)}, \ldots, r_n\},$$

a set of production parameters describing the energy (p_j's), material (q_j's), and signal (r_j's) flows output from the entity. Again,

$$P_{i,(j-1)} = \{P_{i,j}\} \cup \{a_{j-1}, b_{j-1}, c_{j-1}\} \in S_{i,(j-1)},$$

for $j = 2, 3, \ldots, n$, and $P_{i,0} = P_{i,1}$.

$M = \{S_{i',(j+1)}\}$, a singleton set that holds the entity symbol from the next higher level of abstraction that is instantiated in whole or in part by $S_{i,j}$.

K = a set of knowledge parameters about $S_{i,j}$ relevant to evaluating its performance.

The string grammar representation of a tensile testing machine on the mth level of abstraction is $S_{w,m}S_{x,m}S_{y,m}S_{z,m}$. Table 1 displays the values of the activation and production parameters on the mth level of abstraction for the entities in the string ($i = w, x, y, z$).

4.2. Grammar Rules

The rules of the abstraction grammar combine entities of a given level of abstraction, j, into a feasible design, using the design from level $j+1$ as a pattern. To assure that the designs are feasible, the rules must impose restrictions upon the joining of function and form entities (indicated by adjacency in the symbol string), so that generated strings represent valid function structures or feasible machine component assemblies. Physical principles exist which govern the assembly of machine components to enable proper functioning. For example, assembling a gear to a rod for the transfer of rotational energy will only work if the gear is axially mounted on the rod or if the rod is grooved to mesh with the gear teeth; gear teeth

Table 1. Entity parameters for function structure of Fig. 2(d).

Entity name	*Activation parameters on abstraction level m*			*Production parameters on abstraction level m*		
	a_m	b_m	c_m	p_m	q_m	r_m
$S_{w,m}$	Load energy	Specimen	Activation signal	Tensile force	Specimen	Activation signal
$S_{x,m}$	Tensile force	Specimen	Activation signal	Tensile force	Deformed specimen	Specimen data
$S_{y,m}$	Null*	Null	Specimen data	Null	Null	Force measure
$S_{z,m}$	Null	Null	Specimen data	Null	Null	Deformation measure

*Note. The Null parameter values signify that the entity accepts any parameter or flow type and outputs it without a change.

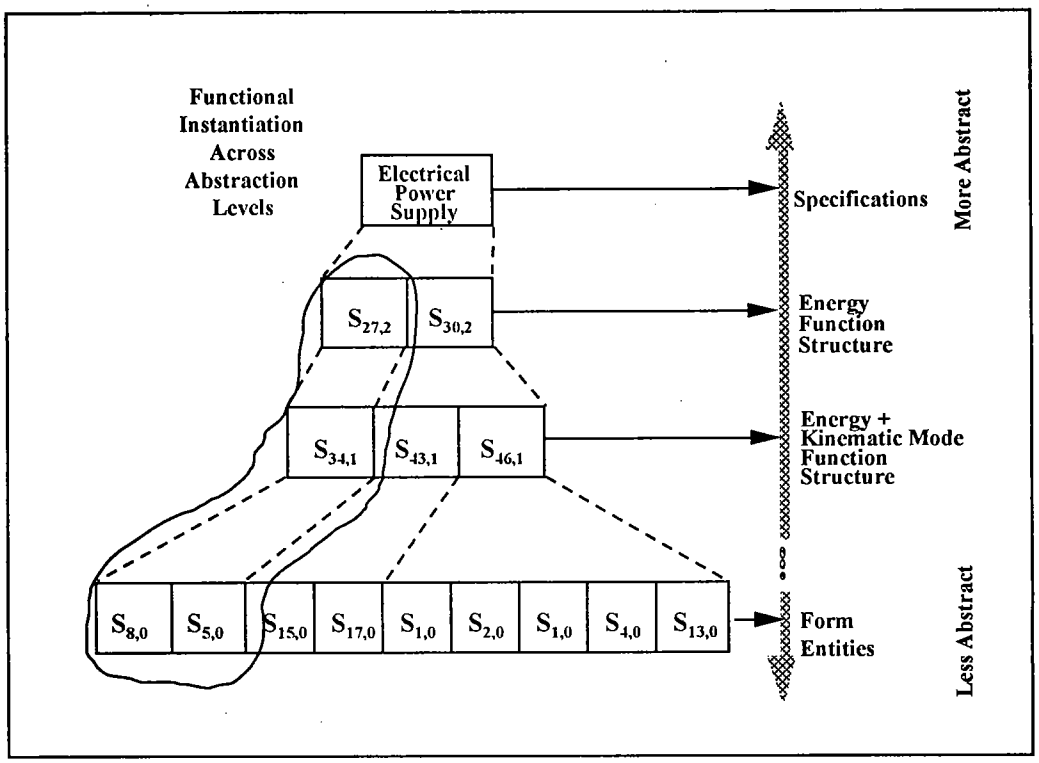

Fig. 3. Illustration of design across level of abstraction.

contacting the surface of a smooth rod will not, in principle, transfer motion to the rod. Compatibility requirements of the same kind are present when joining the functional abstractions of the machine components into function structures. In this grammar implementation, for valid function entity joining, the grammar rules must require that the flows between entities be of the same or compatible types.

The compatible flow representation strategy finds natural expression in the context-sensitive rules for the string grammar. These rules apply on each level of abstraction, using the entities of that level. A typical rule schema on level m of the abstraction grammar is the following:

$$S_{w,m}X_m \rightarrow S_{w,m}S_{x,m}X_m,$$

where $\exists A_{x,m} \subset S_{x,m}$ such that $A_{x,m} = P_{w,m} \subset S_{w,m}$.

Here, X_m is a non-terminal symbol of the grammar which cannot appear in a final string. $S_{w,m}$ and $S_{x,m}$ are terminal symbols and represent function structure entities on level m. If the above rule schema were *not* context-sensitive, the presence of the entity $S_{x,m}$ would have no impact on how X_m was converted to a new substring and the rule application could be written as follows:

$$X_m \rightarrow S_{x,m}X_m.$$

Rules which reference the activation and production parameter sets of the entities are the means for ensuring the creation of only feasible machine designs. Using designs as patterns across the levels of abstraction also relies on referencing parameters of the entities of interest.

$$X_{(m-1)}S_{w,m} \rightarrow S_{t,(m-1)}X_{(m-1)}S_{w,m}$$

where $\exists A_{t,m} \subset S_{t,(m-1)}$ such that $A_{t,m} = A_{w,m} \subset S_{w,m}$.

In this rule schema example, design is occurring on the $(m-1)$ level of abstraction and entities that fulfill the functionality of entity $S_{w,m}$ are added to the string.

Rule schemas for the implemented abstraction grammar are given in the Appendix, but we can illustrate the instantiation of designs across levels of abstraction using an actual electrical power supply design (Fig. 3) generated by FFREADA from the entity library introduced in Table 2. In this set of designs, one on each level of abstraction, the first energy level entity, $S_{27,2}$, representing the conversion of rotational energy to translational energy, is instantiated to entity $S_{34,1}$, representing the conversion of *reciprocating* rotational energy to *reciprocating* translational energy, which is in turn instantiated to the string $S_{8,0}S_{5,0}$, representing an assembly of a torsional spring connected to a pinion and rack. This

Table 2. FFREADA's entity library.

ID #	Entity name	Power conversion efficiency	Cost	Activation energy parameters		Production energy parameters	
				a_2	a_1	p_2	p_1
1	gear pair	0.98	1	R	CRe	R	CRe
2	belt drive	0.85	1	R	CRe	R	CRe
3	power screw	0.9	1	R	CRe	T	CRe
4	shaft	1	1	R	CRe	R	CRe
5	rack & pinion	0.9	1	R	Re	T	Re
6	flywheel	0.8	1	R	Re	R	Re
7	cam	0.95	1	R	C	T	Re
8	torsional spring	0.9	1	R	Re	R	Re
9	electric gen (B)	0.9	1	R	CRe	E	C
10	beam & crank	0.93	1	T	C	R	Re
11	coil spring	0.9	1	T	Re	T	Re
12	electric motor	0.9	1	E	CRe	R	CRe
13	electric gen (A)	0.9	1	R	CRe	E	CRe
14	solenoid	0.95	1	E	C	T	Re
15	lever	1	1	T	Re	T	Re
16	pulley system	0.9	1	T	C	T	C
17	linkage	0.93	1	T	Re	R	CRe
18	transformer (A)	0.93	1	E	C	E	C
19	piezo generator	0.2	1	T	Re	E	Re
20	diode	0.9	1	E	CRe	E	C
21	half-wave rectifier	0.5	1	E	Re	E	C
22	full-wave rectifier	1	1	E	Re	E	C
23	ratchet (A)	0.85	1	T	Re	R	C
24	ratchet (B)	0.85	1	T	Re	R	Re
25	ratchet (C)	0.85	1	T	C	R	C
26	ratchet (D)	0.85	1	T	CRe	R	C
27	R → T	N/A	N/A	R	N/A	T	N/A
28	R → E	N/A	N/A	R	N/A	E	N/A
29	T → R	N/A	N/A	T	N/A	R	N/A
30	T → E	N/A	N/A	T	N/A	E	N/A
31	E → R	N/A	N/A	E	N/A	R	N/A
32	E → T	N/A	N/A	E	N/A	T	N/A
33	R → T CRe → CRe	N/A	N/A	R	CRe	T	CRe
34	R → T Re → Re	N/A	N/A	R	Re	T	Re
35	R → T C → Re	N/A	N/A	R	C	T	Re
36	T → R C → Re	N/A	N/A	T	C	R	Re
37	E → R C → CRe	N/A	N/A	E	C	R	CRe
38	E → T C → Re	N/A	N/A	E	C	T	Re
39	T → R CRe → Re	N/A	N/A	T	CRe	R	Re
40	T → R Re → C	N/A	N/A	T	Re	R	C
41	T → T C → Re	N/A	N/A	T	C	T	Re
42	T → R C → C	N/A	N/A	T	C	R	C
43	T → R Re → CRe	N/A	N/A	T	Re	R	CRe
44	T → E Re → C	N/A	N/A	T	Re	E	C
45	E → T CRe → CRe	N/A	N/A	E	CRe	T	CRe
46	R → E CRe → C	N/A	N/A	R	CRe	E	C
47	T → E Re → Re	N/A	N/A	T	Re	E	Re
48	transformer (B)	0.85	1	E	C	E	C

assembly performs the function of converting reciprocating rotational energy into reciprocating translational energy. A complete derivation of this power supply design appears in Fig. 4.

The abstraction grammar admits all possible feasible designs from the available entity libraries. For example, in the power supply design, entity $S_{34,1}$ could also be instantiated by a single component that fulfills the functional requirement, say $S_{5,0}$, a pinion and rack. $S_{34,1}$ could also be instantiated by an essentially

Generation of a Power Supply Design

Level 2 of Abstraction

Begin: I

After R_1: $X_2\alpha_2\alpha_1\pi_2\pi_1$,

With $\alpha_2 = R$, $\alpha_1 = Re$, $\pi_2 = E$, $\pi_1 = C$

After R_2: $S_{27,2}X_2\alpha_1\pi_2\pi_1$

After R_3: $S_{27,2}S_{30,2}X_2\alpha_1\pi_2\pi_1$

After R_4: $S_{27,2}S_{30,2}X_2\alpha_1\pi_1$

After R_5: $X_1\alpha_1\pi_1S_{27,2}S_{30,2}$

Level 1 of Abstraction:

After R_2: $S_{34,1}X_1\pi_1S_{27,2}S_{30,2}$

After R_4: $S_{34,1}S_{27,2}X_1\pi_1S_{30,2}$

After R_2: $S_{34,1}S_{27,2}S_{43,1}X_1\pi_1S_{30,2}$

After R_3: $S_{34,1}S_{27,2}S_{43,1}S_{46,1}X_1\pi_1S_{30,2}$

After R_4: $S_{34,1}S_{27,2}S_{43,1}S_{46,1}S_{30,2}X_1$

After R_5: $X_0S_{34,1}S_{27,2}S_{43,1}S_{46,1}S_{30,2}$

Level 0 of Abstraction:

After R_2: $S_{8,0}X_0S_{34,1}S_{27,2}S_{43,1}S_{46,1}S_{30,2}$

After R_3: $S_{8,0}S_{5,0}X_0S_{34,1}S_{27,2}S_{43,1}S_{46,1}S_{30,2}$

After R_4: $S_{8,0}S_{5,0}S_{34,1}S_{27,2}X_0S_{43,1}S_{46,1}S_{30,2}$

After R_2: $S_{8,0}S_{5,0}S_{34,1}S_{27,2}S_{15,0}X_0S_{43,1}S_{46,1}S_{30,2}$

After R_3: $S_{8,0}S_{5,0}S_{34,1}S_{27,2}S_{15,0}S_{17,0}X_0S_{43,1}S_{46,1}S_{30,2}$

After R_4: $S_{8,0}S_{5,0}S_{34,1}S_{27,2}S_{15,0}S_{17,0}S_{43,1}X_0S_{46,1}S_{30,2}$

After R_2: $S_{8,0}S_{5,0}S_{34,1}S_{27,2}S_{15,0}S_{17,0}S_{43,1}S_{1,0}X_0S_{46,1}S_{30,2}$

After R_3: $S_{8,0}S_{5,0}S_{34,1}S_{27,2}S_{15,0}S_{17,0}S_{43,1}S_{1,0}S_{2,0}X_0S_{46,1}S_{30,2}$

After R_3: $S_{8,0}S_{5,0}S_{34,1}S_{27,2}S_{15,0}S_{17,0}S_{43,1}S_{1,0}S_{2,0}S_{1,0}X_0S_{46,1}S_{30,2}$

After R_3: $S_{8,0}S_{5,0}S_{34,1}S_{27,2}S_{15,0}S_{17,0}S_{43,1}S_{1,0}S_{2,0}S_{1,0}S_{4,0}X_0S_{46,1}S_{30,2}$

After R_3: $S_{8,0}S_{5,0}S_{34,1}S_{27,2}S_{15,0}S_{17,0}S_{43,1}S_{1,0}S_{2,0}S_{1,0}S_{4,0}S_{13,0}X_0S_{46,1}S_{30,2}$

After R_4: $S_{8,0}S_{5,0}S_{34,1}S_{27,2}S_{15,0}S_{17,0}S_{43,1}S_{1,0}S_{2,0}S_{1,0}S_{4,0}S_{13,0}S_{46,1}S_{30,2}X_0$

After R_5: $S_{8,0}S_{5,0}S_{34,1}S_{27,2}S_{15,0}S_{17,0}S_{43,1}S_{1,0}S_{2,0}S_{1,0}S_{4,0}S_{13,0}S_{46,1}S_{30,2}$

Complete Design

Specifications: $R \rightarrow E$ $Re \rightarrow C$

Level 2 Design: $S_{27,2}S_{30,2}$

Level 1 Design: $S_{34,1}S_{43,1}S_{46,1}$

Level 0 Design: $S_{8,0}S_{5,0}S_{15,0}S_{17,0}S_{1,0}S_{2,0}S_{1,0}S_{4,0}S_{13,0}$

Fig. 4. Machine design generation.

infinite number of different assemblies that each begin with a torsional spring and end in a pinion and rack component or by other assemblies with the same functionality. Since the design generation process is not deterministic and the number of feasible designs is infinite, we must combine it with a tool to explore a limited number of diverse design solutions, evaluating each and ultimately converging to the best possible design. Simulated annealing is an optimization tool that leads to good solutions without an exhaustive search of all possible combinations. A form of simulated annealing that we call *recursive annealing* provides us with a means for both design generation control and optimization.

5. FFREADA: Function to Form REcursive Annealing Design Algorithm

The proposed iterative design model is implemented by a recursive simulated annealing algorithm called FFREADA. To use FFREADA the designer instantiates the abstraction continuum by selecting levels of abstraction to be used, creates the entity library, establishes and inputs initial specifications and creates the objective function used in the evaluation of the instantiated designs on the form level.

5.1. Simulated Annealing in FFREADA

Simulated annealing is a procedure for locating a globally optimal solution by systematically exploring solution states using a controlled Monte Carlo algorithm (Kirkpatrick *et al.* 1983). The algorithm generates design states and then evaluates each state, always accepting a better state and accepting an inferior state according to a probability based upon a *metropolis* algorithm (Metropolis *et al.* 1953). The acceptance of inferior states decreases over time, according to a parameter called the annealing temperature, allowing the algorithm to converge to an optimal solution.

Conventional simulated annealing is guaranteed to converge to a globally optimal solution only under rigorous algorithmic conditions. We make no optimal convergence claims regarding our recursive annealing. The algorithm traverses the design space, converging on a good but not necessarily optimal solution. The annealing schedule FFREADA uses is a modified version of the adaptive schedule suggested by Huang *et al.* (1986). Annealing on a level of abstraction is considered to be converged when a number of successive rejections or a minimum temperature is reached.

FFREADA uses simulated annealing in two ways. Simulated annealing controls the process of instantiating a function structure on the second lowest level of abstraction to the form level. This ensures that the best possible instantiation emerges after each visit to the form level of abstraction. Simulated annealing is also used recursively between levels of abstraction to propagate design evaluation feedback from the best possible instantiation on the form level of abstraction to each higher functional level of abstraction used in the design process. Accordingly, design decisions are made concerning the changing function structures at each level of abstraction.

5.2. FFREADA

The following pseudo-code illustrates FFREADA's control structure. The array, $T[j]$, holds the values of the annealing temperature for each level of abstraction. The level's current temperature value determines the probability of accepting a worse design solution and decreases during operation of the algorithm.

This control program assigns the black box specifications of a desired machine to the variable initial_design and begins the design process by calling the function FFREADA:

```
begin
    level = top_level_of_abstraction
    initial_design = specs
                    /* specs are the initial black box
                     specifications of the machine */
    design_solution = FFREADA(initial_design,
                                level − 1, specs)
end
```

FFREADA is the recursive annealing control program. FFREADA controls the design process across the levels of abstraction, the recursive annealing process on the form level of abstraction (level 0) and the recursive annealing occurring between levels of abstraction:

```
FFREADA(design@level_+_1, level, specs)
begin
  new_design = generate(level,
                        design@level_+_1, specs)
  evaluation = infinity
  T[level] = initial
while(T[level] > final or not converged){
                                /* annealing loop */
  new_design = perturb(design, level, specs)
  if(level = 0)
                        /* design at the form level */
  new_design_eval = objective_function(new_design)
else    /* design on the next lower level is required
              before evaluation can take place */
  new_design_solution = FFREADA(new_design,
    level − 1, specs)
  new_design_eval = value(new_design_solution)
              /* returns value of the new design */
  if metrop(new_design_eval, evaluation) = accept
                        /* metropolis algorithm */
  design = new_design_solution
  evaluation = new_design_eval
reduce T[level]
  }
  return design
end
```

Generate is a recursive function that produces a design on one abstraction level below that of the pattern design by instantiating the entities of the pattern design one by one (but not necessarily one *to* one) on the next lower level of abstraction with the function *instantiate*. Generate calls itself recursively after each successive entity in the pattern is instantiated. Generate is also used to reinstantiate one or more entities of a pattern design during the perturbation process.

The function *perturb* allows FFREADA to make small changes, or perturbations, in designs. It is a requirement of the simulated annealing procedure to explore designs similar to designs accepted during the annealing process. Perturb randomly selects an entity in a design and reinstantiates it. This will require subsequent redesigns on all levels below the pattern level.

5.3. Implemented Abstraction Grammar Details

We have implemented an abstraction grammar similar to the adapted Pahl and Beitz energy, material and signal flow representation discussed in Section 4, using only the energy flow through entities. We follow the description of spatial grammars found in Krishnamurti and Stouffs (1993) and formalize our abstraction grammar (introduced in Schmidt and Cagan 1993) as a string grammar, examining the ramifications of this characterization in Section 5.3.3.

A string grammar, G, is the four-tuple, $G = (N, T, R, I)$, where:

N = a set of non-terminal symbols, those which appear in a string only during its creation and do not appear in final strings;

T = a set of terminal symbols, the symbols which must comprise all final strings;

R = rules for creating new strings from old; and

I = an initial string.

The strings are created from the set of vocabulary elements, V, where $V = N \cup T$, but $N \cap T = \varnothing$, and $T \subset U$. U is the power set of strings that can be created from members of T, and U includes ε, the empty string. The language of strings generated by the grammar is $L(G) = v \mid v \in T^*\}$, where T^* is the least set of terminal vocabulary elements closed under string concatenation *and* the rules of the grammar. Typical transformations on strings, including addition, subtraction and the substring relation, are outlined by Krishnamurti and Stouffs (1993).

In our grammar, a completed string represents a machine design. Consider the power supply design shown in Fig. 3. (FFREADA's library of entities used to create the design is listed in Table 2, shown in Section 4.) It can also be represented by the following string of entities:

$$S_{8,0}S_{5,0}S_{34,1}S_{27,2}S_{15,0}S_{17,0}S_{43,1}S_{1,0}S_{2,0}$$
$$S_{1,0}S_{4,0}S_{13,0}S_{46,1}S_{30,2}.$$

In this single string, the tree-like structure of the design is flattened out into functional groupings. Consider the substring $S_{8,0}S_{5,0}S_{34,1}S_{27,2}$. In this substring, the function represented by entity $S_{27,2}$ is satisfied by entity $S_{34,1}$. Entity $S_{27,2}$ represents the conversion of rotational energy to translational energy and is the pattern entity for instantiation on level 1 of abstraction. The function represented by $S_{34,1}$ is the conversion of rotational energy available on a reciprocating basis to translational energy also on a reciprocating basis and satisfies the pattern given by entity $S_{27,2}$. The substring $S_{8,0}S_{5,0}$ represents an assembly of a torsional spring connected to a pinion and rack and is the physical instantiation of the functioning required by the substring's pattern entity, $S_{34,1}$. The functional phrasing of the strings generated by the proposed grammar is commented upon in Section 5.3.3 and further illustrated in Fig. 5.

In terms of our string grammar formalism, we interpret the grammar symbols as follows:

$G = (N, T, I, R)$,
where

$N = \{X_j, \alpha_m \pi_m\}$, with $j = 0, 1, \ldots, n$ and $m = 1, 2, \ldots, n$. In this set of non-terminal symbols, the "X_j" symbols are markers for string creation on the "jth" level of abstraction. Also included in the set of non-terminals are parameter symbols α_m and π_m. These parameters are from the original specifications for the machine to be designed. The meaning of these parameters is defined by FFREADA's implementation for a particular problem. Here we use them to indicate activation (α_m's) and production energy (π_m's) parameters as described in the definitions of Section 5.3.1.

$T = \{S_{i,j} \in E_j, j = 0, 1, \ldots, n$, the set of all entities on level "j" of abstraction$\}$. In this implementation, the subscript "i" is an entity library identification number. In the entity library listed in Section 4, $E_2 = \{S_{27,2}, S_{28,2}, S_{29,2}, S_{30,2}, S_{31,2}, S_{32,2}\}$.

$I = \alpha_n \alpha_{(n-1)} \cdots \alpha_1 \pi_n \pi_{(n-1)} \cdots \pi_1$, an initial string consisting of the input (α_j) and output (π_j) specification parameters for the machine. The length of the substring of α_j's must equal that of the π_j's and must be equal to n, the number of levels of abstraction above the 0th level. There are no

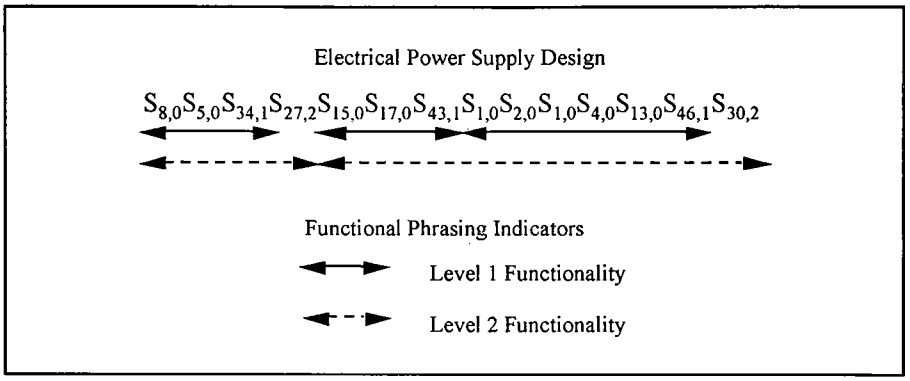

Fig. 5. Functional segmentation of designs.

additional energy parameters associated with the 0th level, beyond those given on levels 1 through n.

$R = \{R_1, R_2, \ldots, R_m\}$, a set of m rules for transforming an existing string to a new string. Each $R_m = (a, b)$, where the pair a and b are valid strings under the grammar. The string a is the left-hand side of the rule $a \rightarrow b$. As such, the string a must contain at least one member of the set N, the non-terminal symbols of the vocabulary.

5.3.1. Abstraction Grammar Entities "$S_{i,j}$"

The abstraction grammar uses a library of entities, each entity denoted by "$S_{i,j}$". A sample entity library used for the generation of power supplies is given in Table 2 in Section 4. The entity $S_{i,j}$ represents a set of information described as follows:

$S_{i,j}$[3] $= \{A_{i,j}, P_{i,j}, M, K\}$, an entity on the "$j$th" level of abstraction is represented by "$S_{i,j}$". The subscript "i" is an entity identification number. The subscript "j" is the level of abstraction of the entity.

$A_{i,j} = \{a_j, a_{(j+1)}, \ldots, a_n\}$, a set of activation energy parameters belonging to entity $S_{i,j}$, each describing the class of activation energy of entity "i". The parameter subscript indicates the level of abstraction on which that parameter becomes known. Parameters are defined for levels j, $(j+1), \ldots, n$, for $j > 0$. As a result, $A_{i(j-1)} = \{A_{i,j}\} \cup \{a_{j-1}\} \in S_{i,(j-1)}$, for $j = 2, 3, \ldots, n$, and $A_{i,0} = A_{i,1}$.

In this implemented abstraction grammar, machines, machine components and entities are represented according to the type of energy flowing through them during activation. This representation is derived from the Pahl and Beitz (1988) approach used in developing their systematic design method. Here, three working levels of abstraction are used. They are:

- Level 3 = specifications.
- Level 2 = energy function structure;
  energy parameters a_2, $p_2 \in \{R, T, E\}$ where: "R" = rotational; "T" = translational; and "E" = electrical.
- Level 1 = kinematic-mode-plus-energy function structure;
  kinematic-mode parameters a_1, $p_1 \in \{C, Re, CRe\}$ where: "C" = continuous;[4] "Re" = reciprocating; and "CRe" = both continuous and reciprocating modes are possible and either will activate the entity.
- Level 0 = machine component layout (i.e. form level).

$P_{i,j} = \{p_j, p_{(j+1)}, \ldots, p_n\}$, a set of produced energy parameters, each describing the class of energy produced (output) from entity $S_{i,j}$. Again,

$$P_{j-1} = \{P_j\} \cup \{p_{j-1},\} \in S_{i,(j-1)},$$

for $j = 2, 3, \ldots, n$, and $P_{i,0} = P_{i,1}$.

$M = \{S_{i',(j+1)}\}$, a singleton set of an entity at the next higher level of abstraction that is the pattern for the selection of entity $S_{i,j}$ on the jth level.

$K = \{k_1, k_2, \ldots, k_m\}$, a set of attribute information for entity $S_{i,j}$. In our current implementation of the grammar, k_1 is the entity name, k_2 is the entity cost, and k_3 is the entity's power conversion efficiency. Entity cost and power conversion efficiency are only applicable to component entities.

It should be clear that the sets $A_{i,j}$ and $P_{i,j}$ are not unique to entity $S_{i,j}$. Other entities may have the same energy parameter sets. In fact, successful operation of the grammar relies upon the fact that there are entities with similar activation and production parameters.

[3] This is a slight change from earlier work (Schmidt and Cagan 1993) where the "j" was the number of the entity in the string representation of a machine. In a string grammar, the number identifying the order of the entity in a string is not part of the representation, but rather is retrieved using a function. For example, in the string $n = S_{2,j}S_{5,j}S_{1,j}S_{3,j}S_{4,j}$, $f(S_{3,j}) = (S_{3,j}, 4)$.

[4] When used to modify electrical emergy flow, "continuous" refers to direct current and "reciprocating" indicates alternating current.

5.3.2. Abstraction Grammar Rule Schemas

The abstraction grammar rules consist of generation rules, i.e. those that generate a new design from a set of specifications, and perturbation rules, i.e., those that modify an existing design. During FFREADA's design generation, rule application begins by randomly selecting the entity $S_{x,j}$ from the library. If the entity is of the proper type and the rule requirements are satisfied, the rule is applied to the design. Otherwise a new entity is selected and tested with the rule. A demonstration of rule applications to generate the design of an idealized power supply design is given in Fig. 4, illustrating the following grammar rules:

R1: *initial rule* – converts the initial string, I, into machine specifications and beginning design marker, X_n.

R2: *satisfying activation parameters* – adds an entity to the design on level j. This entity must satisfy the activation parameters of the pattern design entity on level $(j+1)$. If $j = n$ the pattern is the set of machine specification parameters.

R3: *Adding additional entities to satisfy a functional pattern* – adds additional entities to the existing design on level j to satisfy production parameters of the pattern entity on level $(j+1)$. If $j = n$ then the pattern is the set of machine specification parameters.

R4: *satisfying function pattern parameters* – compares current design parameters on level j for satisfaction of pattern design entity on level $(j+1)$. If $j = n$ the pattern is the set of machine specification parameters.

R5: *design termination* – ends the design process on level j and begins the process on level $(j-1)$. If $j = 0$ the design generation is finished.

Generation rule schemas are defined in the Appendix, as are perturbation rules.

5.3.3. Observations

Describing the abstraction grammar with a string grammar formalism allows us to make three important observations. First, a review of a final machine design description makes obvious the serial nature of the designs. The representation of the form and function entities combined with the grammar rules limit the designs to serial arrangements of components. While it is clear that other, more descriptive, grammars are necessary to design more complex machines, the class of machines we call serial machines contains interesting examples for study (e.g., power transmission machines, gear trains, copiers). In these types of machines, the behavior of the machine tends to be dominated by the process the machine performs. Our formalism guarantees that our grammar will be able to generate all serial machines possible from the library entities.

Second, since our implementation applies rule R4 (recognizing function pattern satisfaction) whenever possible, the grammar is biased toward the generation of the shortest string of entities that satisfy the specifications.

Third, the most useful observation proceeding from the formalism as a string grammar is the functionally segmented character of the designs produced by the grammar. This is evident in the segmentation of the power supply design of Fig. 3 illustrated in Fig. 5. There are two immediate consequences of this functional segmentation.

- Explicit function sharing is not possible.
  Function sharing is a desirable property (Ulrich and Seering 1988) and the inability to directly impose it via rule application is a shortcoming of this grammar. However, indirect function sharing, where a more complex function structure on Level 1 or Level 2 replaces a longer function structure, is possible with FFREADA and occurs during the annealing process. It would also be possible to create a function sharing operator like that of Ulrich and Seering and apply it after the design generation process or as an option during perturbation.
- Generalization of performance metrics to functions is possible.
  The segmented nature of the designs allows us to abstract performance values attributed to specific components back to the function structures which they satisfy. In this manner we can begin to articulate performance metrics that a designer can use at functional levels of abstraction.

6. Power Supply Design Generation with FFREADA

The success of personal portable electronic devices such as cassette tape and compact disc players prompts speculation on the feasibility of generating power from the physical activity that occurs during walking or running. Two possible motion sources are considered here: the motion of the leg at the knee, characterized as reciprocating rotational energy, and the energy of the heel compression during each step, characterized as reciprocating translational energy. Rotational energy is a common input to the power generation process, translational energy is not. Technology in the form of piezoelectric polymer transducers makes the small compression available