Research in Engineering Design (1997) 9: 195–213
© 1997 Springer-Verlag London Limited

Research in
Engineering
Design

GGREADA: A Graph Grammar-Based Machine Design Algorithm

Linda C. Schmidt,[1] and Jonathan Cagan[2]

[1] Department of Mechanical Engineering, University of Maryland, College Park, MD, USA; [2] Mechanical Engineering Department, Carnegie Mellon University, Pittsburgh, PA, USA

Abstract. *A class of graph grammar-based design algorithms for the generation of near-optimal machine designs is proposed. The GGREADA algorithm is implemented to design scale model carts from Meccano® Set components. A graph grammar for the carts is defined. GGREADA successfully designs minimum-weight carts that provide specified load-bed areas. GGREADA's exploration capability is also demonstrated by creating probability frequency plots of likely cart surface areas and weights from random samples taken from the space of all possible cart designs. The advantages that a designer may gain from grammar-based design tools are discussed along with the representational challenges that remain in exploiting grammars for this purpose.*

Keywords. Configuration design; Generative design; Grammar-based design; Machine design

1. Introduction

Research into computational mechanical design algorithms has as its goal the production of tools that create optimal or optimally-directed [1] designs. Grammars are a means to generate a space of alternatives to a design problem. Their use to implement *generate and optimize*-style mechanical design algorithms is now being explored. The generate and optimize approach to the mechanical design problem involves the application first of a method to generate and evaluate feasible designs, then the application of an optimization technique to select the best design. Mechanical design problems resist solution in this manner because it is computationally difficult to generate and evaluate a complete set of feasible alternatives in response to a solution description.

The concept of an optimizing grammar-based designer assistance algorithm was proposed in Schmidt and Cagan [18] along with a recursive model of design. This model presents design as a set of hierarchically ordered individual design processes occurring at different levels of abstraction. The search for a good design is controlled by a recursive simulated annealing process [12]. The initial work by Schmidt and Cagan presented a grammar-based algorithm named FFREADA and applied it to a conceptual design problem. The name 'FFREADA' stands for function-to-form recursive annealing design algorithm. The proposed class of grammar-based algorithms features an abstraction grammar[1] to describe and generate a space of design alternatives, a library of machine components and their functional interpretations used to create designs of machines at different levels of abstraction, and a hierarchical optimizing search technique to select the best design from the space of feasible designs. FFREADA uses a string abstraction grammar to generate a space of highly-idealized power supply designs from a library of rudimentary mechanical components and their abstracted functional descriptions. Each design is created by grammar rules which transform specifications into a string of symbols through a series of predefined design processes. Each design process uses the symbols of one level of abstraction and creates a design string representing the machine on that level of abstraction. The symbols in the string represent abstract descriptions of machine functions ordered with symbols of components that perform those functions.

FFREADA uses a different component library to generate and explore a space of hand-held power drill designs in Schmidt and Cagan [19]. This

Correspondence and offprint requests to: Linda C. Schmidt, Mechanical Engineering Department, University of Maryland. College Park, MD 20742-3035, USA. E-mail: lschmidt@eng.umd.edu; Phone: +1 301 405 0417; fax: +1 301 314-9477.

[1] An *abstraction grammar* is a grammar that generates a set of designs of a single device, one design on each of a number of hierarchically ordered levels of abstraction.

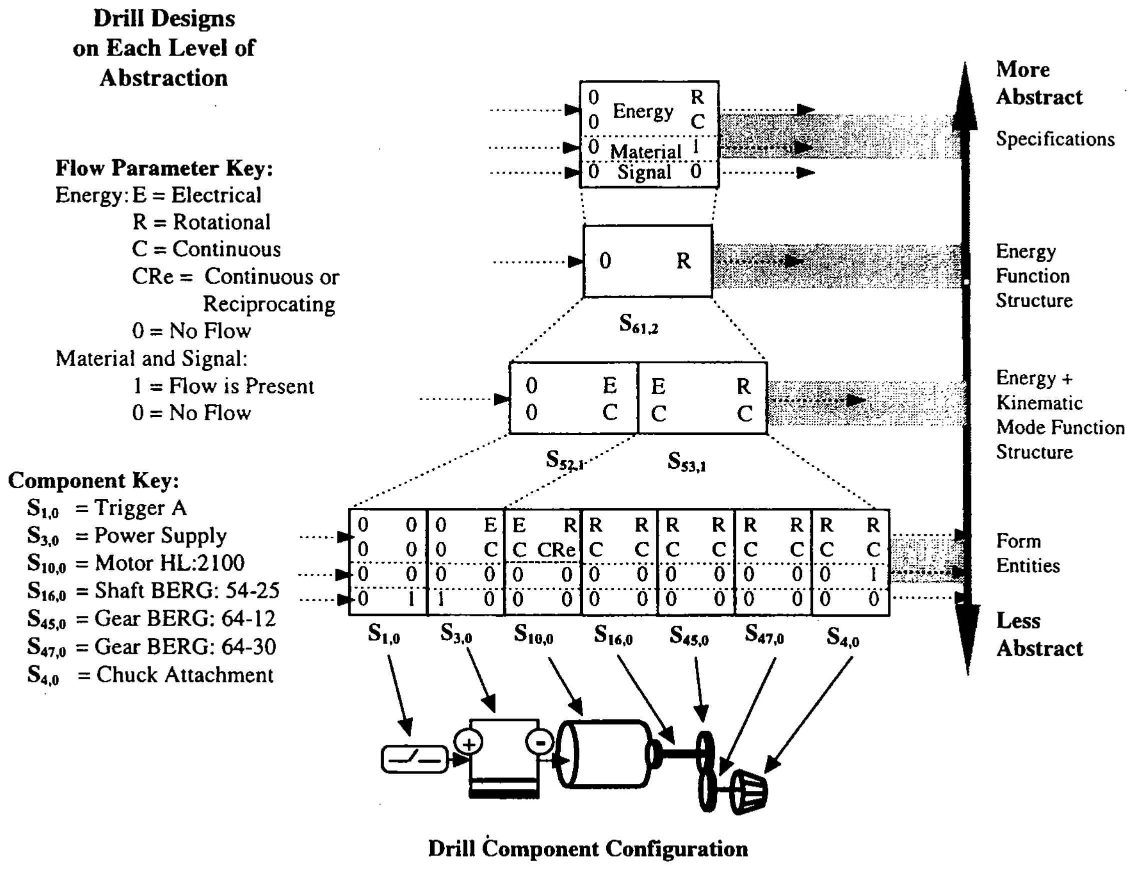


Fig. 1. An optimal drill design by FFREADA.

second FFREADA implementation demonstrates the approach on configuration and catalog selection design tasks. An optimal, 1.36 N m (1 ft-lb) torque drill design solution (i.e., the lowest-cost drill assembled from a catalog of components that delivers 1.36 N m of torque) generated by FFREADA's string grammar is shown in Fig. 1. The string of symbols representing functions and forms is depicted as a tree along the abstraction continuum to emphasize the hierarchical nature of the design process. A diagram of the final drill configuration appears under the form level design.

One clear shortcoming of a string abstraction grammar implementation is its restriction to generating string-like arrangements of components. Not all mechanical devices are arrangements of components in series. A bicycle is a mechanical device that has some simple component assemblies (e.g., chain drive, hand brake) combined in a complex configuration with wheels, seat, steering, brake, and structural support systems. A flat-bed truck or horse-driven cart is another general mechanical machine that is made up of many component assemblies in a non-serial arrangement. An automobile carburettor has many complex connections and physical and functional relationships between subassemblies and individual component parts. Consider also the standard nail clipper, made from four basic components. The clipper design is considered elegant with structural parts also supplying the functionality for the cutting procedure [22]. Design grammars able to express a wide range of mechanical configurations are needed if grammar-based designer tools are to become a reality.

This paper builds upon the previous work on string abstraction grammar design algorithms by introducting a class of graph abstraction grammar algorithms. The graph grammar allows representation of non-serial component arrangements and functional relationships. Also presented here is an implemented graph grammar algorithm named GGREADA, an acronym for graph grammar recursive annealing design algorithm. As an example, GGREADA is used to design rolling carts using scale model engineering components from Meccano® Erector® Sets. This is a combination of configuration and catalog selection design tasks. GGREADA supports a basic level of function sharing, defined by Ulrich and Seering [22] to be 'the simultaneous implementation of several functions with a single structural element'. Prior work demonstrates the applicability of the recursive design model and the grammar-based, generate and optimize approach to mechanical design tasks, ranging from conceptual design to catalog selection, for design problems that can be solved with a serial arrangement of machine components. Successful demonstration of graph grammar design algorithms supports the potential for grammar-based, generate and optimize tools for a broader cross-section of engineering design problems.

2. The Mechanical Design Problem and Meccano® Set Carts

The engineering design process converts the concept of a device that satisfies a particular need into physical reality. In mechanical design, the process result is a description of a machine that behaves in a certain fashion to meet or exceed given performance requirements. The portion of the mechanical design process targeted here begins when the need is already articulated into a statement of specifications which includes device inputs and expected outputs. When a mechanical design problem is specified, a space of feasible alternatives is also defined, although individual designs are not explicitly articulated. If a technique for generating the space of feasible designs is available, and if the resulting space can be efficiently searched by recovering and evaluating designs within its boundaries, a generate and optimize strategy can be employed to create a designer assistance tool.

2.1. Mechanical Design as State Space Search

The space of design alternatives applicable to a typical machine design problem consists of an infinite number of discrete states limited only by the number and type of components that the designer considers. Each state located in the space represents a new design which may or may not have a performance rating close to the rating of its neighbour states. The design alternative space is combinatorially explosive and ill-behaved. Powerful optimization techniques and search strategies are needed to perform efficient search in such spaces. Our experience with the recursive design model [18] indicates that the recursive simulated annealing strategy used in FFREADA is capable of performing search under these conditions.

To generate a space of machine designs, a process is needed to transform a functional description of machine performance into a group of machine components and a description of their arrangement or 'configuration'. This function-to-form transformation is only accomplished by reasoning about the ability of components to satisfy functional requirements. An abstraction grammar for design uses grammar rules to create designs first at a high level of abstraction, where reasoning is based on functions, and then on more detailed function and form levels of abstraction where the grammar rules select and order components into a machine configuration using higher level functional designs as patterns.

To demonstrate the ability of a graph abstraction grammar-based design algorithm to perform design, we must select a mechanical design problem of sufficient complexity to benefit from graph-like component configuration representation. Since the development of an intelligent functional reasoning engine is beyond the scope of this work and tangential to the concept of grammar-based design, the problem must be of sufficient functional clarity to allow design by rudimentary functional reasoning. The design of a simple cart is selected to demonstrate grammar-based machine design.

2.2. Meccano® Erector® Set Cart Design

Consider creating a designer assistance tool to solve a simple mechanical problem, the design of a rolling cart from a set of Meccano® Erector® Set components (hereafter referred to as Meccano Set pieces or components). Mecanno Set pieces are scale model mechanical device components with realistic action. A machine designed with the Meccano Set grammar is a functioning model of a real-world mechanical device and can be quickly built to articulate designs.

The cart design problem is both a conceptual and a configuration design problem. The design tool must develop alternative concepts which can vary in approaches to the solution, select Meccano Set components to implement the solution concept and determine how the pieces can be arranged into a functioning cart. The designer assistance tool we envisage must be able to select the best possible design from all those it can create. This adds the task of optimization to the design process.

In the cart design implementation, an unlimited number of 12 different Meccano Set components are available to the designer and are shown in Fig. 2, items (a)–(l). These components and their design features are listed in Tables A.1 and A.2 in the Appendix. Items (m) and (n) of Fig. 2 are diagrams of two sub-assemblies that can be used to mount cart

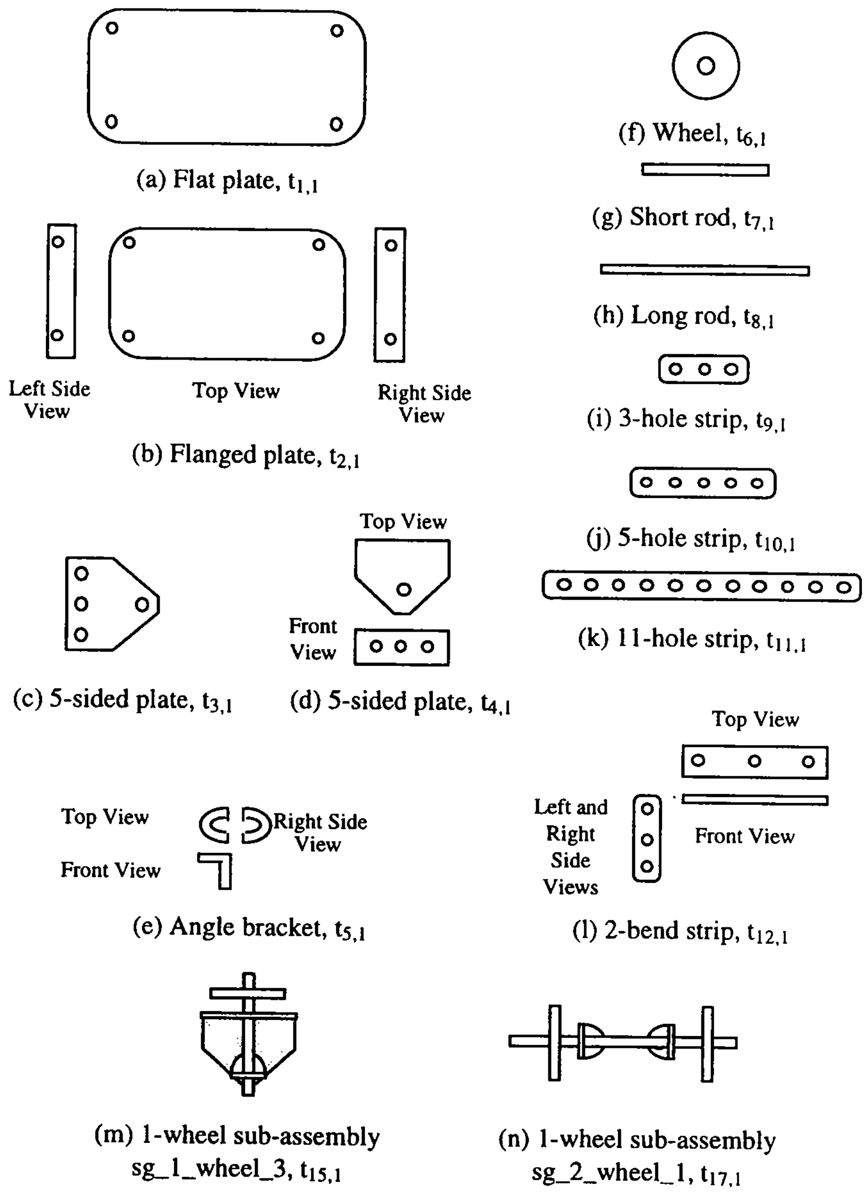

(a) Flat plate, $t_{1,1}$
Left Side View
Top View
Right Side View
(b) Flanged plate, $t_{2,1}$
(c) 5-sided plate, $t_{3,1}$
Top View
Front View
(d) 5-sided plate, $t_{4,1}$
Top View
Right Side View
Front View
(e) Angle bracket, $t_{5,1}$
(f) Wheel, $t_{6,1}$
(g) Short rod, $t_{7,1}$
(h) Long rod, $t_{8,1}$
(i) 3-hole strip, $t_{9,1}$
(j) 5-hole strip, $t_{10,1}$
(k) 11-hole strip, $t_{11,1}$
Top View
Left and Right Side Views
Front View
(l) 2-bend strip, $t_{12,1}$
(m) 1-wheel sub-assembly sg_1_wheel_3, $t_{15,1}$
(n) 1-wheel sub-assembly sg_2_wheel_1, $t_{17,1}$

Fig. 2. Meccano Set cart components and two wheel sub-assemblies.

wheels. There are two additional wheel mounting sub-assemblies allowed in the design process and each of the four sub-assemblies can mount either one or two wheels on any component that has two mounting joints (i.e., all components except the rods and the wheels). A variety of carts can be created from the set of available components. Three different carts are displayed in Fig. 3.

A cart is a rolling platform that can provide surface area for the placement of a load. The amount of surface area available for hauling can be viewed as a design constraint. The design task is described as follows:

Create the capacity for rolling motion by mounting at least three wheels on a base structure and provide a specified amount of horizontal surface area to support a load.

It is clear from Fig. 3 that even our limited number of component types can be used to create many different cart configurations. Using a surface area requirement as a constraint reduces the number of feasible designs for a specific cart design problem but another measure of cart performance must be selected to discriminate between feasible designs. The measure chosen for this exploration is total cart weight. The 'best' cart will be the minimum weight cart that

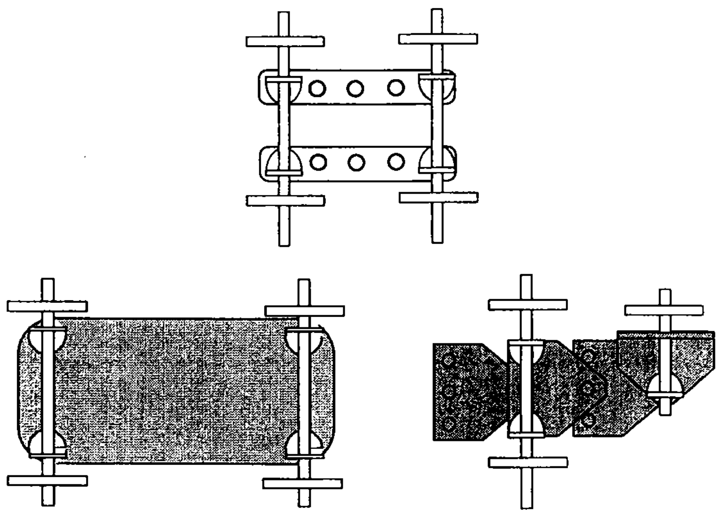


Fig. 3. Possible Meccano Set carts.

provides the requisite surface area and the rolling motion.

3. Grammars and Graph Grammars in Mechanical Design

A grammar is a mathematical formalism useful for mechanical design. The opportunity exists to determine the properties of any language of designs generated by a grammar. For example, a context-sensitive grammar, the type used for design with FFREADA and GGREADA, is closed under the operations of union and concatenation [10]. This is of interest to the researcher combining two grammars, consider merging a grammar that defines a language of gear trains and one that defines a language of clutches. Mullins and Rinderle [14] and Rinderle [17] select grammars for the benefits of their formality and for their inherent transformational nature through the design of bridges and booms, respectively.

Grammars model design as a linguistic formalism, allowing design to be viewed as a domain independent activity and positing interesting research hypotheses. Fitzhorn [6] sets forth a domain-independent, computational theory of design, drawing heavily on the characterization of design as a linguistic process consisting of the manipulation of symbols with requisite translation into artifacts. A Turing machine definition of design is developed in which grammars provide the alphabet for the Turing machine. Fitzhorn proposed that by defining design with a grammar-based formalism, one is able to more fully investigate notions of design complexity and design as state-space search.

Graph grammars have been applied to a variety of design applications. These applications involve graph representations at many different levels of detail, from solid models and features, to components and their behaviors, to complete devices. Fitzhorn [5] proposes a graph grammar to describe physical solids. In work on mechanical feature design, Pinilla et al. [16] use a graph grammar to describe shape features such as notches and perpendicular faces. Fu et al. [7] also present a graph grammar for the representation and transformation of form primitives and machinable features and to propagate constraints. Hoover and Rinderle [9] present a technique to transform a graph-based representation of specifications into an arrangement of physical components for single-speed mechanical power transmissions. Another graph-based representation presented by Sthanusubramonian et al. [20] creates bond graphs of power path topologies and provides transformations into physical components. Finger and Rinderle [4] propose a bond graph grammar, a graph-based language for mechanical design, and apply it to gear box design. Deng [3] uses a graph grammar to design electric motors.

4. GGREADA

GGREADA uses a graph-based abstraction grammar to create designs and a recursive simulated annealing process to select a near-optimal design from those created. Like FFREADA, GGREADA's abstraction grammar generates designs on each of '*m*'

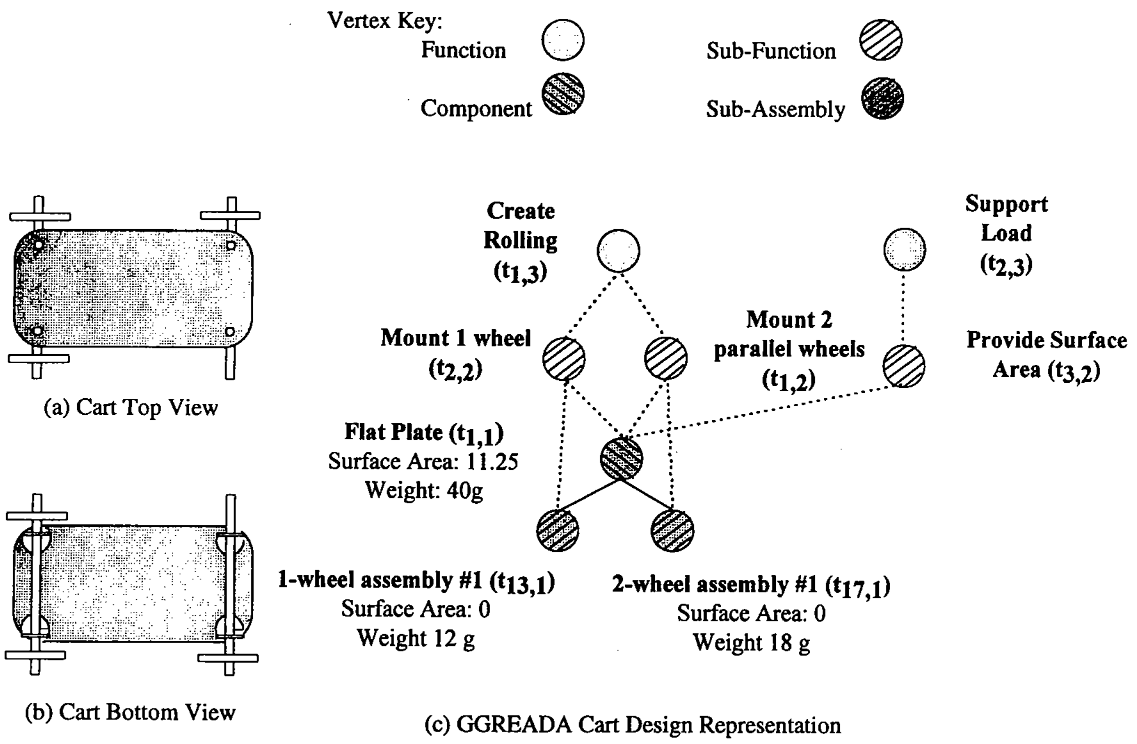

Vertex Key:
Function
Sub-Function
Component
Sub-Assembly
(a) Cart Top View
(b) Cart Bottom View
Create Rolling ($t_{1,3}$)
Support Load ($t_{2,3}$)
Mount 1 wheel ($t_{2,2}$)
Mount 2 parallel wheels ($t_{1,2}$)
Provide Surface Area ($t_{3,2}$)
Flat Plate ($t_{1,1}$) Surface Area: 11.25 Weight: 40g
1-wheel assembly #1 ($t_{13,1}$) Surface Area: 0 Weight 12 g
2-wheel assembly #1 ($t_{17,1}$) Surface Area: 0 Weight 18 g
(c) GGREADA Cart Design Representation

Fig. 4. A typical GGREADA Meccano Set cart design.

levels of abstraction from a pattern design generated on the previous and more abstract level. Unlike FFREADA, GGREADA's abstraction grammar is graph-based. As a result, GGREADA's form level designs have more complex topologies than the power supply and drill designs down by FFREADA. Figure 4 displays a simple cart designed by GGREADA and its representation as a graph by the abstraction grammar. This section begins with a general description of the graph grammar and includes comments on the representation system used and the format of the grammar rules. GGREADA's design model is described next, followed by an introduction to the recursive simulated annealing strategy guiding the design generation process.

4.1. Graph Grammars for Design

A graph grammar is a mathematical formalism for manipulating symbols representing graph vertices and edges. Grammar rules assign function and form entities to vertices and connect vertices with edges. The edges of a graph signify the relationship between two vertices. In GGREADA's grammar, a completed graph expresses a design. A graph grammar articulates an entire language of designs. In this implementation it is a language of carts made from Meccano Set components. Changing an entity available to the grammar as a vertex or revising a rule in the grammar modifies the language of carts.

Each vertex in a completed graph represents a machine function entity or a Meccano Set component entity, also called a form entity. Tables A.1 and A.2 in the Appendix, list all entities available for assignment to a vertex. The graph edges represent either a physical relationship between Meccano Set component entities, a functional relationship between two related function entities, or between Meccano Set components and the function entities that they fulfil. A physical edge between two components indicates that they are connected. Functional relationships are a bit more complex.

Functional relationships exist between entities on different levels of abstraction. An entity is only selected for a design if it fully or partially satisfies the functionality required by another entity in the design pattern from the next higher level of abstraction. For example, in the cart of Fig. 4, two sub-function entities, 'mount one wheel' and 'mount two parallel wheels', each partially satisfy the function 'create rolling' and are connected to the 'create rolling' vertex by a functional edge.

An advantage of graph abstraction grammar is its ability to explicitly represent function sharing. In

Fig. 4's cart, the 'flat plate' component is satisfying the functionality required by all three of the sub-function components. It acts as a mounting piece for the wheel sub-assemblies and it provides surface area for the cart. The functional relationships between the plate and the sub-functions are indicated by graph edges.

4.1.1. Graph Grammar Formalism

A graph grammar, G, is formally represented as follows:

$$\text{Graph grammar:} \quad G = (V_{\text{n}}, V_{\text{t}}, P, S) \tag{1}$$

$$\text{Vertex sets:} \quad V_{\text{n}} = n_{\phi,j}, j = 1, 2, \ldots, m \tag{2}$$

$$V_{\text{t}} = \{t_{i,j}\}, i = 1, 2, \ldots, q_j, \quad j = 1, 2, \ldots, m \tag{3}$$

$$\text{Rule set:} \quad P = \{R_i\} \tag{4}$$

$$\text{Start symbol:} \quad S = n_{\Theta}. \tag{5}$$

Graph grammar G is a collection of non-terminal symbols (vertex set V_n), terminal symbols (vertex set V_{t}), which are assigned a function or form entity by the action of the rule set, rules (P) and a start symbol (S). The non-terminal vertices of a graph (2) indicate where the design is incomplete. Levels of abstraction numbered from 1 to m, are identified in the grammar. Entities representing machine functions and forms ($t_{i,j}$) are defined for each of the m levels of abstraction and numbered from 1 to q_j. For any graph, g_a, the set of non-terminal symbols it contains at any time is $V_{\text{n}}(g_{\text{a}})$. The set of terminal symbols it contains at any time is $V_{\text{t}}(g_{\text{a}})$. Graph grammar rules transform a graph from one configuration to another.

$$\text{Grammar rule:} \quad R_i\colon g_{\text{a}} \rightarrow g_{\text{b}} \quad g_{\text{a}} \text{ containing at least one member of } V_{\text{n}} \tag{6}$$

A graph is typically described as a set of vertices and edges connecting them. Any graph, g_{a}, generated by a graph grammar under this notation can be represented as follows:

$$\text{A graph:} \quad g_{\text{a}} = (V, E) \tag{7}$$

$$\text{Graph vertices:} \quad V = (V_{\text{n}} \cup V_{\text{t}}) \tag{8}$$

$$\text{Graph edges:} \quad E = (E_{\text{f}} \cup E_{\text{p}}) \tag{9}$$

$$E_{\text{f}} = \text{set of functional edges} \tag{10}$$

$$E_{\text{p}} = \text{set of physical edges} \tag{11}$$

$$E \subseteq (V \times V) \tag{12}$$

Using this notation and the cart graph in Fig. 4(c) as graph g_{a}, all the following are true statements:

$$\begin{aligned} g_{\text{a}} = (V, E) = (&\{(t_{1,3}), (t_{2,3}); (t_{2,2}), (t_{1,2}), (t_{3,2}) \\ &(t_{1,1}), (t_{13,1}), (t_{17;1})\} \\ &\{(t_{1,3}, t_{2,2}), (t_{1.3}; t_{1,2}), (t_{2,3}, t_{3,2}) \\ &(t_{2,2}, t_{13,1}), (t_{2.2}; t_{1,1}), (t_{1,2}, t_{1,1}) \\ &(t_{1,2}, t_{17,1}), (t_{3.2}, t_{1,1})(t_{1,1}, t_{13,1}) \\ &(t_{1,1}, t_{17,1})\} \end{aligned} \tag{13}$$

$$t_{17,1} \in V_{\text{t}}(g_{\text{a}}) \tag{14}$$

$$(t_{2,2}, t_{1,1}) \in E_{\text{f}}(g_{\text{a}}) \tag{15}$$

$$(t_{1,3}, t_{2,2}) \in E_{\text{f}}(g_{\text{a}}) \tag{16}$$

$$(t_{1,1}, t_{13,1}) \in E_{\text{p}}(g_{\text{a}}) \tag{17}$$

If u and v are vertices of the graph, a physical edge between them would be represented as (u, v), with (u, v) equivalent to (v, u). Functional edges will exist between vertices representing two machine functions (16) and between a machine function vertex and a form vertex (15) if that form was selected in the design process to satisfy part or all of the function.

4.2.1. Entity Representation System

The entity representation defining components here is the minimum functional representation required to demonstrate the use of GGREADA on the cart design problem. GGREADA's entities are assigned to graph terminal symbol vertices by the action of the rules. The entities are represented by the terminal vertex symbol $t_{i,j}$. Form entities are the Meccano Set pieces depicted in Fig. 2 with their terminal vertex symbols. GGREADA's entity library includes four types of entities: high-level functions, sub-functions, Meccano Set components, and Meccano Set component sub-assemblies. The corresponding four vertex types are indicated in Fig. 4 by different shadings in the graph diagram. The component sub-assemblies are themselves subgraphs but they are treated as a single vertex in the current GGREADA implementation.

In FFREADA's representation system, library entities are characterised as transformers of energy, signal, and material. Pattern matching between entity flows governs the joining of the entities in designs. FFREADA's energy characterization works well for machine design problems when the purpose of the machine is to change the nature of the energy flowing through the machine and where machines comprise serial arrangements of components. In the cart design problem, energy flow through components does not capture the essence of the component behavior in carts. This GGREADA implementation focuses on the behaviors of the components and how they can be

Table 1. Vertex entity knowledge

Level	Knowledge item	Description
Level 3: function entity	Name	Function's name
	Function code	Identifying number assigned to each function or sub-function in GGREADA's entity library
Level 2: Sub-function entity	Name	Sub-function's name
	Function code	Identifying number assigned to each function or sub-function in GGREADA's entity library
	Function applicability codes	List of function codes for which the sub-function can contribute some satisfying behavior (i.e., 'mount 1 wheel' contributes to 'create rolling')
Level 1: Component piece and sub-graph	Name	Component or sub-graph name
	Function applicability codes	List of function codes for which the entity can contribute some satisfying behavior
	Weight	Weight of the component or the subassembly in grams
	Surface area	Horizontal surface area of the component or subassembly
	Number of joints	Number of joints on the component or subassembly available for making physical connections

assembled to exploit some of those behaviors to meet the functionality required by a cart. Behavior is defined as all a component can do, while function is the behavior for which a component is selected. For example, in the cart of Fig. 4, the 'flat plate' is selected by the grammar to function as a base piece for the mounting of wheels. However, the 'flat plate' also provides surface area to support a load and this capability will be recognized by the rules of the grammar later in the design process.

In an ideal Meccano Set graph grammar, all behaviors derivable from the set of components would be catloged, leading to a greater number of generalized sub-functions and high-level functions. In this prototype Meccano Set graph grammar implementation, three sub-functions are used: 'mount one wheel', 'mount two parallel wheels', and 'provide surface area'. Two high-level functions are used: 'create rolling' and 'provide surface area'. Each is assigned a function code. All components that can provide some of the functionality are assigned a function applicability code that matches the appropriate function code. The rules of the grammar choose components and component sub-assemblies by matching function codes. The function codes and function applicability codes for each entity in GGREADA's library are listed in the tables in the Appendix. Each vertex entity also has knowledge information characteristic of the function or form entity it represents. This information is described in Table 1.

4.1.3. Grammar Rules

The rules of a graph grammar transform a graph, g_a, which includes at least one non-terminal vertex, into a new graph, g_b (6). The grammar's rules systematically proceed from the initial symbol (n_Θ) to a complete cart design. The design generation rules express the conditions required to add a new vertex and its assigned function or form entity to an existing graph. GGREADA's grammar rules are defined in the Appendix.

Figure 5 describes one rule and its application during a cart design process. The rules make use of a set of functions, called characteristic functions, that take one or two vertices as arguments and evaluate functional or physical relationships in existence in the graph. Several characteristic functions are used in articulating rule R_{10} described in Fig. 5. The functions are described in the Appendix and their values for rule R_{10} are illustrated here.

$$F_s(t_{x,2}) = 0 \tag{18}$$

$$F_{fe}(t_{y,1}, t_{x,2}) = 1 \tag{19}$$

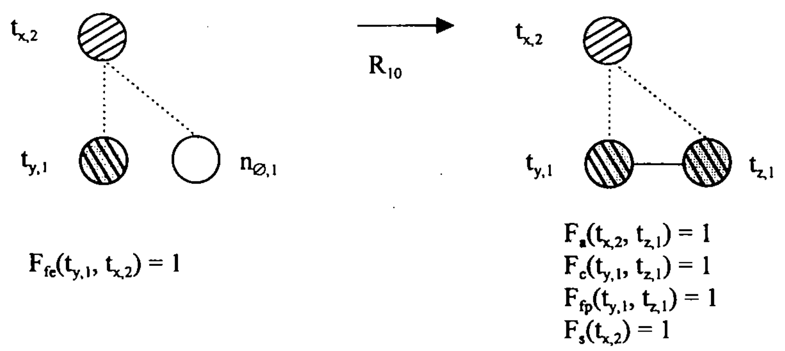
$$F_a(t_{z,1}, t_{x,1}) = 1 \tag{20}$$

$$F_c(t_{z,1}, t_{y,1}) = 1 \tag{21}$$

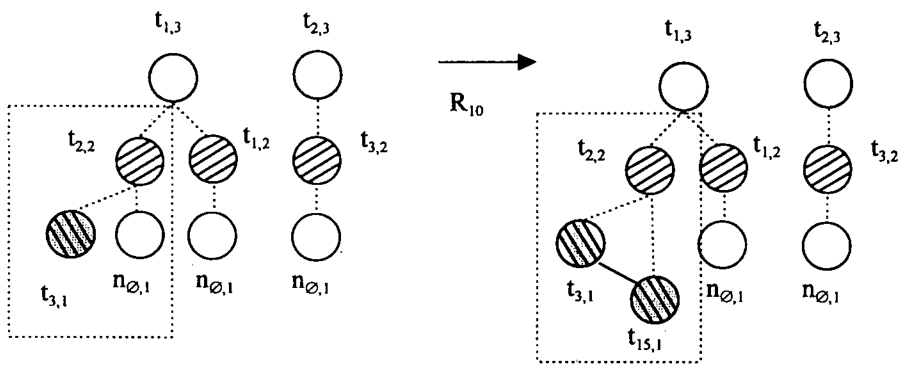
$$F_{pe}(t_{y,1}, t_{z,1}) = 1 \tag{22}$$

$$F_s(t_{x,2}) = 1 \tag{23}$$

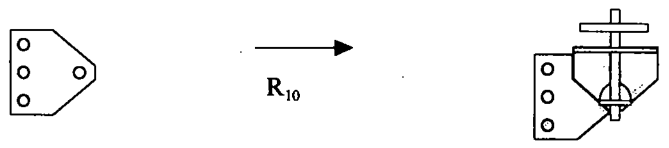
Figure 5(a) expresses the form of the general rule. The left-hand side of the rule expresses the condition that there exists a sub-function entity, $t_{x,2}$, with unsatisfied functionality (18) and a form level entity, $t_{y,1}$, that satisfies its functionality (19) but not completely. The right-hand side of the rule expresses the conditions of the graph after the rule is applied. Now included in the graph is a form level entity $t_{z,1}$ that has


$t_{x,2}$
$t_{y,1}$
$n_{\varnothing,1}$
R_{10}
$t_{z,1}$
$F_{fe}(t_{y,1}, t_{x,2}) = 1$
$F_a(t_{x,2}, t_{z,1}) = 1$
$F_c(t_{y,1}, t_{z,1}) = 1$
$F_{fp}(t_{y,1}, t_{z,1}) = 1$
$F_s(t_{x,2}) = 1$

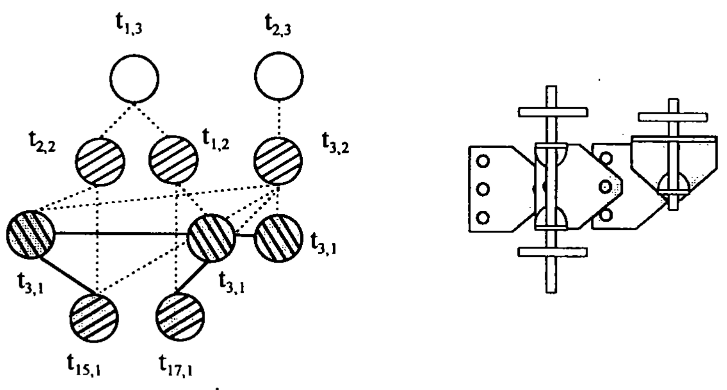
(a) Rule R_{10}: Form Addition and Attachment with Function Satisfaction.



(b) Graph view of R_{10} application during the design process.



(c) Component view of R_{10} application during the design process.



(d) Graph and component representation of complete design

Fig. 5. A graph grammar rule and its application.

functionality required by sub-function entity $t_{x,2}$ (20). It is physically possible to connect form entity $t_{z,1}$ to another form entity in the graph, $t_{y,1}$ (21) and this connection is made (22) By becoming part of the graph, $t_{z,1}$ has fully satisfied the functionality of entity $t_{x,2}$ (23) causing the elimination of the non-terminal vertex, $n_{\phi,1}$. Figure 5(b) depicts a specific graph before and after the application of rule R_{10} with the affected portion of the graph outlined. A view of the Meccano Set pieces in the cart design as the rule is applied is shown in Fig. 5(c) and the complete graph and cart design are shown in Fig. 5(d).

The amount of geometric knowledge available to GGREADA is minimal. The rules assure that the connection between the Meccano Set pieces can be made because the appropriate number of joints are open but they do not specify which joints are used. Full connectivity information is determined by GGREADA, yet some interpretation of results by the designer is required to assemble the final designs.

4.2. GGREADA's Recursive Design Model

GGREADA follows the recursive design model in which design occurs systematically on hierarchically ordered levels of abstraction [18]. GGREADA's design process is decomposed into design at an appropriate number of levels of abstraction. Each level of abstraction has a set of entities that represent machine functions or machine forms. Figure 6 diagrams GGREADA as implemented for the Meccano Set cart design problem. Other implementations of the algorithm can be written with different abstraction level and annealing process configurations. Neither the recursive design model nor the abstraction grammar algorithms limit the design process to a particular number or type of levels of abstraction. It is assumed that the designer selects the number and defining characteristics of each level of abstraction. In theory, FFREADA or GGREADA can be implemented with an infinite number of levels of abstraction. The only requirement is that the levels are hierarchically ordered and the recursive design model is still valid.

GGREADA's design process applies a graph abstraction grammar to the design problem. The algorithm systematically creates a design on a level of abstraction from a pattern design from the next higher level. Design on the highest level of abstraction uses the specifications as the pattern for the process. The rules of the grammar assure that the completed design will be functionally valid. An element of randomness is purposely introduced into the design

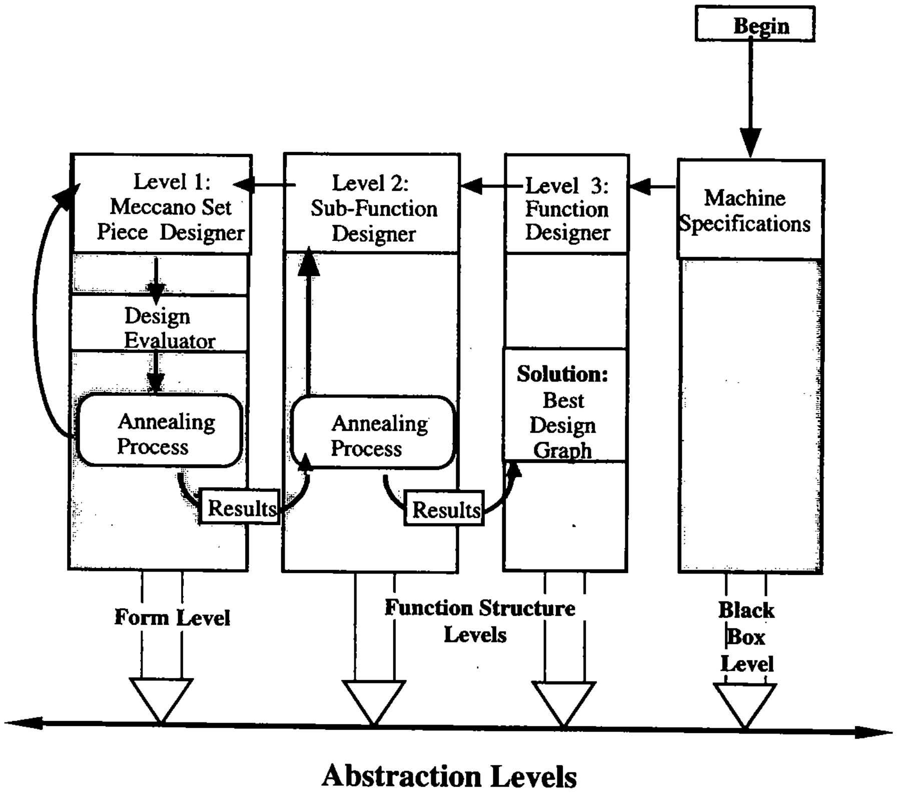

Begin
Level 1: Meccano Set Piece Designer
Level 2: Sub-Function Designer
Level 3: Function Designer
Machine Specifications
Design Evaluator
Annealing Process
Annealing Process
Solution: Best Design Graph
Results
Results
Form Level
Function Structure Levels
Black Box Level
Abstraction Levels

Fig. 6. GGREADA implemented to design Meccano Set carts.

process by selecting grammar rules randomly and applying them only if they are valid at that particular point in the design process. This allows GGREADA to generate any number of valid designs from a library of entities.

To converge to good cart designs, GGREADA applies simulated annealing on two levels of abstraction below the machine specifications level. These are the 'sub-function' level and the 'Meccano Set piece' or 'form' levels of abstraction. Simulated annealing is applied in a recursive fashion to control the design generation process on the sub-function and Meccano Set component (form) levels of abstraction. In the current implementation, there is only one design possible on the function level of abstraction, Level 3, so no optimization process is necessary at that level of design. The only Level 3 design generated by the grammar is to assign a vertex to 'create rolling' and another to 'support load'. This is accomplished by rule R_1 described in Table A.3 in the Appendix.

4.3. GGREADA's Simulated Annealing Optimization Strategy

GGREADA's graph abstraction grammar creates a cart design. By incorporating a stochastic rule selection mechanism into the grammar, it is possible to automatically generate any number of cart designs. Each design generated can be thought of as being randomly selected from the language of cart designs described by the Meccano Set cart grammar. GGREADA can be operated in a random design generation mode to simulate the results of a random sampling of all designs possible from the space of designs expressed by the grammar. Statistics that characterise designs representative of the design space can be collected from the sample of designs. Observing that GGREADA's design mechanism generates individual states within a space of designs, it is natural to integrate a state-space search technique into the design process. Simulated annealing directs the search through the design space to end on the state that represents a design that is optimal, or near-optimal, in terms of quantifiable design and performance attributes.

Designs develop from choices made at different levels of abstraction that are eventually instantiated into Meccano Set components on the form level of abstraction. To make intelligent choices about alternatives at a high level of abstraction is difficult because machine performance cannot be predicted with great accuracy until a complete physical design candidate has been specified. Designing with an abstraction grammar allows the choice of design alternatives to be made on each level of abstraction and each alternative becomes one state in a simulated annealing process on that level of abstraction. The value of that state is the value of the best design that can be generated from the grammar using that state's design as a pattern for completing the design process. To find this best design value, a complete simulated annealing process is conducted, using the design grammar to generate alternative designs from the pattern representing the design state under investigation. This sets up the need for another simulated annealing process on the next lower level of abstraction to evaluate the alternative design states and use those values in the first annealing process. A recursive process continues until the design states being generated exist on the form level of abstraction where they can be evaluated directly using performance characteristics. For details on the recursive annealing strategy see Schmidt and Cagan [18].

5. Cart Design Results

Like many conceptual and configuration design problems, the cart design problem is highly combinatorial and even a small set of components creates a vast space of feasible design alternatives. To derive the size of the design state space, consider first the number of options for mounting wheels to satisfy the 'create rolling' function described by cart specifications. There are 32 unique ways to mount either 1 or 2 wheels (8 base pieces with 2 mounting joints and 4 sub-assemblies for each mounting scenario). There are 4 different ways in which the 'mounting' components can be combined to produce the mounting of 3 wheels, which is all that is required to satisfy the create rolling function. This results in 67 584 different cart designs (see Table 2) before a base piece is selected for the mounting and before the

Table 2. Wheel mounting options.

Wheel mounting options	Number of possible designs
Mount 2 parallel wheels + mount 1 wheel	32^2 or 1024
Mount 2 parallel wheels + mount 2 wheel	32^2 or 1024
Mount 2 parallel wheels + mount 1 wheel + mount 1 wheel	32^3 or 32 768
Mount 1 wheel + mount 1 wheel + mount 1 wheel	32^3 or 32 768
Total design options:	67 584

surface area requirement is considered. With no minimum surface area requirement there are $2 \times 67\,584$ designs available from the library, using one Meccano Set base piece (with 4 or more joints). The number of designs expands combinatorially from this point as new pieces are selected for addition to the design 'to provide surface area'. For example, to provide $51.6\,\mathrm{cm}^2$ of surface area, there are over 600 000 designs with 3 base pieces and over 3.6 million designs with 4 base pieces. The design space grows in size as designs using more base pieces are included.

5.1. Random Design Generation Results

GGREADA can generate and evaluate designs with the goal of exploration rather than optimization. Figure 7 presents three probability frequency histograms displaying the likelihood of randomly generating carts constrained to provide a minimum of $12.9\,\mathrm{cm}^2$ ($2\,\mathrm{in}^2$), $25.8\,\mathrm{cm}^2$ ($4\,\mathrm{in}^2$), or $77.4\,\mathrm{cm}^2$ ($12\,\mathrm{in}^2$) of surface area. For each histogram, 200 000 designs were generated in GGREADA's random design generation mode. Figure 7 shows how the design alternatives requiring more surface area are a subset of the problem with the lowest surface area constraint. It also demonstrates that the grammar is responsive to quantifiable constraints. Also interesting in Fig. 7 is the second mode of solutions to the $12.9\,\mathrm{cm}^2$ and $25.8\,\mathrm{cm}^2$ problems that appears at about $52\,\mathrm{cm}^2$. These designs contain the larger Meccano Set component pieces.

Figure 8 presents histograms displaying the probability of randomly generating a cart with a given weight and minima of $12.9\,\mathrm{cm}^2$, $25.8\,\mathrm{cm}^2$, and $77.4\,\mathrm{cm}^2$ of surface area. The profiles of the design state space suggested by these histograms contain many local minima, leading to the conclusion that optimization may be difficult.

5.2. Simulated Annealing Optimization Results

GGREADA's recursive simulated annealing optimization mode is used to find optimal or near optimal solutions to the cart design problems. GGREADA uses an annealing schedule on each level of abstraction that is a simple derivative of the adaptive annealing schedule of Huang et al. [11]. Rather than a probabilistic approach to determining the achievement of equilibrium (i.e. convergence), GGREADA uses a consecutive-rejection strategy with an iteration limit at each temperature. The annealing runs used the following convergence criteria:

Level 3,	Function structure: N/A (deterministic design);
Level 2,	Sub-function structure; and
Level 1,	Meccano Set Component: Earlier of reaching a consecutive rejection limit, or generating 1000 designs at current temperature.

For the $25.8\,\mathrm{cm}^2$ surface area objective design problem, the consecutive rejection limits were 75 and 50 for design on abstraction levels 2 and 1, respectively. For all other design scenarios, the consecutive rejection limits were set at 10 for design on all levels of abstraction. These limits are set as low as possible while still yielding good results. The runs took no longer than 45 s each on the DEC 3000 workstation used for all the computing of this work.

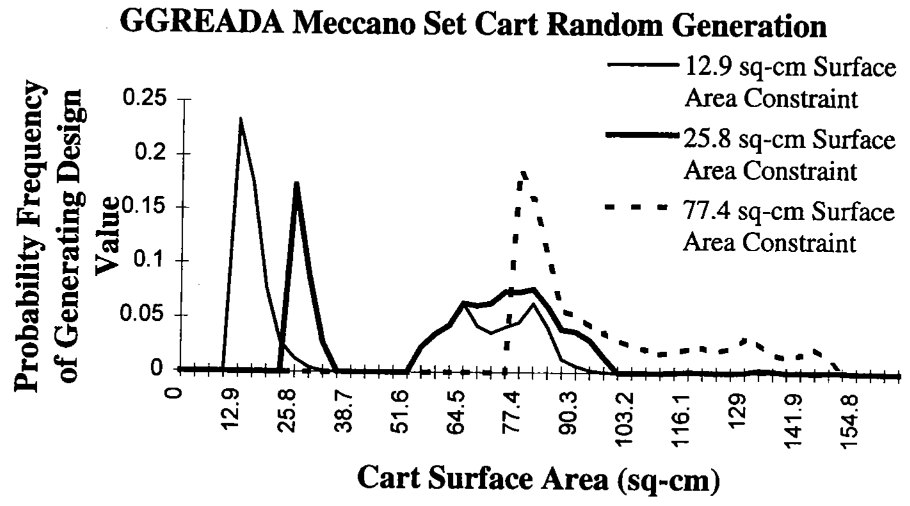


Fig. 7. Histogram of cart surface area during random generation.

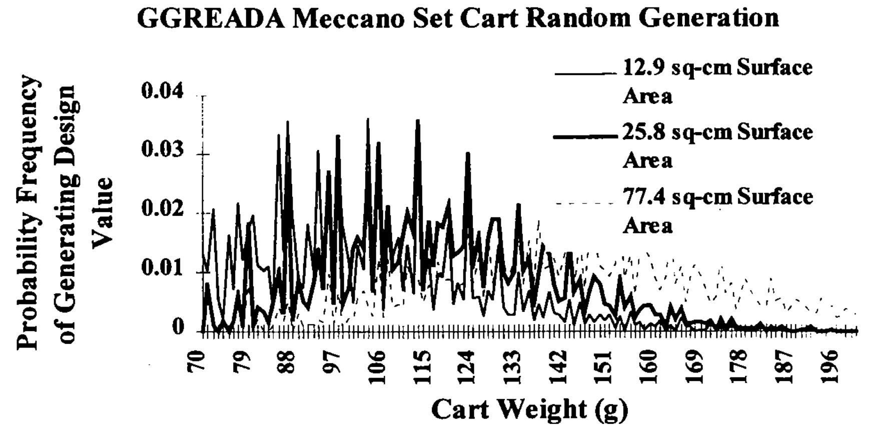


Fig. 8. Histogram of cart weights during random generation.

Carts are designed for minimum weight using two different surface area constraints, 25.8 cm^2 and 77.4 cm^2. The results are summarised in Table 3. In the table, the weight of the design GGREADA converged to is in the column titled 'Converged run result'. The weight of the lightest design found during the run is in the column titled 'Run minimum'. The average weight of all carts designed during the run is in the column titled 'Run average'. Five annealing runs for the cart design problem with a 25.8 cm^2 surface area constraint are listed first in Table 3. GGREADA converges to either a 68-g solution (the cart depicted in Fig. 5) which is the optimal solution, or a 70-g solution (the cart described in Fig. 4). The annealing algorithm responds to different search experiences. In runs 2, 3, and 4, when the 68-g solution is generated, the algorithm searches longer as GGREADA attempts to converge to the run minimum design.

Table 3 continues with runs for the minimum weight cart design problem with a constraint that the cart provides 77.4 cm^2 of surface area for load placement. The 78 g optimal solution is a derivative of flat plate design of Fig. 4 which provides 72.6 cm^2 of surface area. To increase the surface area, a flanged plate wheel mounting sub-assembly is selected (sg_1_wheel_3) and connected to the flat plate on which two parallel wheels are mounted.

GGREADA is also applied to the problem of designing a minimum weight cart with a 12.9 cm^2 load bed and the results are summarised in Table 4. The column headings of the table are interpreted as in Table 3, except that the designs are valued in square inches rather than grams. The optimization process is also successful for this objective function, with GGREADA converging to designs with the minimum 12.9-cm surface area in all cases. There are a number of alternative designs that have 12.9 cm^2 of surface area. One is an 11-hole strip on which a two-bend strip wheel mounting sub-assembly and any other wheel mounting sub-assembly are attached.

Table 3. GGREADA annealing results for cart designs with minimum weight.

Surface area constraint	Run number	Converged run result (g)	Run minimum (g)	Run average (g)	Form level designs (No.)
25.8 cm^2	1	70	70	82.5	8560
	2	68	68	82.5	14 947
	3	68	68	83.3	13 858
	4	70	68	82.3	12 854
	5	70	70	82.5	8460
	Average	69.2	68.8	82.6	11 736
77.4 cm^2	1	78	78	113.5	393
	2	78	78	107.7	764
	3	78	78	115.4	551
	Average	78	78	112.2	569

Table 4. GGREADA annealing results for cart designs with 12.9 cm^2 surface area.

Surface area constraint	Run number	Converged run result (cm^2)	Run minimum (cm^2)	Run average (cm^2)	Form level designs (No.)
12.9 cm^2	1	12.9	12.9	15.10	1216
	2	12.9	12.9	14.71	731
	3	12.9	12.9	14.97	930
	Average	12.9	12.9	14.93	959

The designs generated by GGREADA are feasible within the context of the grammar. There is no explicit geometric reasoning occurring in GGREADA's rule applications. As a result, some interpretation of the joining of the components is necessary to build the designs. Some designs feasible under the grammar may not produce carts with all the characteristics of a good cart. For example, some carts using an 11-hole base piece will not be stable when assembled. If the wheel mounting sub-assemblies do not provide some offset from the base piece (like the five-sided flanged plate sub-assembly wheel mounting option does), the cart may have all wheels aligned on the same axis and tip over. This problem can be rectified by incorporating more sophisticated geometric reasoning either into the grammar rules or by adding stability constraints that would serve to eliminate unstable designs during the search process.

6. Discussion

The cart example presented here demonstrates that a graph grammar-based algorithm can be developed to design near-optimal Meccano Set carts. GGREADA's graph grammar generated a variety of designs spanning the space of design solutions. Function sharing is also supported by the graph grammar representation and allows the grammar rules to recognise and use the explicitly defined behaviors of the cart components. GGREADA's generative capacity is highlighted by the production of probability frequency histograms that document design samples taken from the space articulated by the grammar.

The cart example is just one application of a graph grammar-based design algorithm. This approach can be extended to other applications by writing appropriate graph abstraction grammars. Three advantages can be gained by using grammars for design. They are:

1. The succinct articulation of a complete space of design alternatives and the ability to recover particular designs from the space [8],
2. The transformation of the design process into a computationally direct production system,
3. The ability to recognise new elements of the design by virtue of flexibility in the design representation (i.e., the recognition of emerging design elements).

GGREADA's design results clearly illustrate the usefulness of a grammar as a means of describing a space of designs. The other two advantages following from the use of a grammar for design are addressed in more detail in this discussion.

The advantage of ease of computational implementation is achieved in part by using the grammar to aid in the design of data structures compatible with grammar rules. In the GGREADA algorithm's current implementation, the diagram of a cart graph itself (like the graph in Fig. 4) is a visual representation of the internal data structure created to represent the cart. Imposing this data structure on the design allows a clarity of programming useful in implementing rules. This is especially true when implementing rules that depend on some property of the function or form being assigned to a particular node. For example, in the cart design graph of Fig. 4, the node representing the flat plate points to nodes that represent two different wheel sub-assemblies, allowing a joint availability sub-routine to easily count the number of available joints remaining on the flat plate.

Unlike the first two grammar advantages heading this section, the emergence advantage (3) is one that can be gained from a grammar formalism and not from most other techniques applied to the design process. The ability of a shape grammar to create and represent emerging shapes and to recognise them in a computational implementation has been demonstrated ([2, 13, 21] for example). This suggests that the potential exists to establish a method for representing and recognising emergence in a grammar that represents functions and forms, although the means of accomplishing this task will not be the same as for a shape grammar. In the case of a shape grammar, the subject of interest is the emergent shape, a shape not explicitly represented but made up from a collection of other shape lines. In the case of grammar based machine design, the subject of interest is emergent behavior, behaviors of components that are incidental to the functioning for which they are first included in the design or that occur because a component interacts with another in the design (e.g. resonance due to a vibration). Recognising emerging behavior is important to good mechanical design because it is the basis for implementing fully-realized function sharing.

This GGREADA implementation does not take advantage of the potential for recognizing good designs through emergent behavior because the representation power of the grammar is not fully exploited. Recognising the potential of the full spectrum of behavior of a machine component is a useful and powerful ability for a design algorithm. A design grammar that has the potential to recognise emergent mechanical behaviors would require a representation system that expressed all potential behaviors in a standard fashion. Krishnamurti's [13] maximal line representation for shapes makes it possible to implement routines to recognise emergent shapes. The approach used in maximal line representation for shape grammars suggests the following guidelines for representations conducive to the recognition of emergent behavior in a machine design grammar:

1. Each mechanical component would be represented by a collection of relevant behavior elements, which may not be unique, but, when combined with a form description, would uniquely represent each component.
2. Common behaviors would be represented as a collection of behavior elements.
3. Grammar rules would peruse a design of multiple components, and match behavior elements combining from components to recognise emergent behavior.
4. The behavior representation would be linked to the grammar's structure so that manipulating the structure, without regard to what it represents, will successfully manipulate the design.

There are representation systems existing in the engineering design literature that may provide a starting point for representing and recognising the behavior of components singly and in assemblies. One system is bond graphs, a method of describing physical systems and representing their energy and power flows, and their dynamic behavior [15].

7. Conclusions

The GGREADA algorithm application presented here demonstrates three abilities of a graph grammar design algorithm: the representation of a space of Meccano Set Cart designs; the characterization of the space of designs by taking random surveys of design alternatives and evaluating the sample; and the generation of optimal and near-optimal designs. Three conclusions are drawn from these results, together with previous grammar-based design work. First, the recursive design model can be adapted for use on tasks spanning the complete design problem, from conceptual design, as in the first FFREADA application, to a combination of configuration and catalog selection design as with the FFREADA drill application and the work presented in this paper. Second, in each application we have demonstrated the ability of the generate and optimize approach to the machine design problem to both find optimally-directed designs and to characterise the design state space. Third, we can see the flexibility of the abstraction grammar approach to express a wide range of function and form relationships. Work remains to refine grammars for machine design by developing more sophisticated representations to allow exploitation of the full benefits of a grammar formalism as applied to engineering design.

Acknowledgments

Linda Schmidt gratefully acknowledges the support of the National Defense Science and Engineering Graduate (NDSEG) Fellowship Program. The authors acknowledge the National Science Foundation for providing support for this research under grants DDM-9258090 and DDM-9301096. Both authors also thank Dr. Robert Sturges for helpful discussions on the Meccano Set cart design problem.

References

1. Cagan J, Agogino A. Inducing constraint activity in innovative design. AI EDAM: Artificial Intelligence in Engineering, Design, Analysis and Manufacturing 1991; 5(1): 47–61
2. Chase SC. Shapes and shape grammars: from mathematical model to computer implementation. Environment and Planning B 1989: 16: 215–242
3. Deng Y-S. Feature based design: synthesizing structure from behavior. PhD Dissertation, Department of Industrial Engineering, University of Pittsburgh 1994

4. Finger S, Rinderle J. A transformational approach to mechanical design using a bond graph grammar. Proceedings of the 1st ASME Design Theory and Methodology Conference, Montreal 1989
5. Fitzhorn P. A linguistic formalism for engineering solid modeling. In: Ehrig H, Nag M, Rozenberg G, Rosenfeld A (eds). Graph-grammars and their application to computer science, 3rd International Workshop. Springer-Verlag, New York 1986
6. Fitzhorn P. A computational theory of design. Design Computing 1988; 2 (2): 221–233
7. Fu Z, De Pennington A, Saia A. A graph grammar approach to feature representation and transformation. International Journal of Computer Integrated Manufacturing 1993; 6 (102): 137–151
8. Gips J, Stiny G. Production systems and grammars: a uniform characterization. Environment and Planning B 1980; 7: 399–408
9. Hoover S, Rinderle J. A synthesis strategy for mechanical devices. Research in Engineering Design 1989; 1: 87–103
10. Hopcroft J, Ullman J. Formal languages and their relation to automata. Addison-Wesley, Reading, MA 1969
11. Huang MD, Romeo E, Sangiovanni-Vincentelli A. An efficient general cooling schedule for simulated annealing. Proceedings of the IEEE International Conference on Computer-Aided Design (ICCAD-86), Santa Clara, CA, November 11–13 1986
12. Kirkpatrick S, Gelatt C Jr, Vecchi M. Optimization by simulated annealing. Science 1983; 220 (4598): 671–679
13. Krishnamurti R. The arithmetic of shapes. Environment and Planning B 1980; 7: 463–484
14. Mullins S, Rinderle J. Grammatical approaches to engineering design, Part I: An Introduction and commentary. Research in Engineering Design 1991; 2: 121–135
15. Paynter H. Analysis and design of engineering systems. MIT Press, Cambridge, MA 1961
16. Pinilla J, Finger S, Prinz F. Shape feature description using an augmented topology graph grammar. Preprints: NSF Engineering Design Research Conference, Amherst, MA, June 11–14 1989. pp 285–300
17. Rinderle J. Grammatical approaches to engineering design, Part II: Melding configuration and parametric design using attribute grammars. Research in Engineering Design 1991; 2: 137–146
18. Schmidt L, Cagan J. Recursive annealing: a computational model for machine design. Research in Engineering Design 1995; 7: 102–125
19. Schmidt L, Cagan J. Configuration design: an integrated approach using grammars. To appear in Proceedings of The 1996 ASME Design Engineering Technical Conferences and Computers in Engineering Conference: Design Theory and Methodology Conference, August 18–22, Irvine, CA 1996
20. Sthanusubramonian T, Finger S, Rinderle J. A transformational approach to configuration design. Proceedings of the 1992 NSF Design and Manufacturing Systems Conference, Atlanta, GA, January 8–10, 1992. pp 419–424
21. Stiny G. Formal devices for design in Newsome SL, Spillers WR and Finger S (eds). Design theory 88, Springer-Verlag, New York 1989
22. Ulrich K, Seering W. Function sharing in mechanical design. AAAI 88: Proceedings of the 7th National Conference on Artificial Intelligence, St. Paul, MN, Vol 1. 1988. pp 342–347

Appendix

This Appendix includes details of GGREADA's Meccano Set cart grammar. Tables A.1 and A.2 list all entities available for assignment to a graph vertex in a graph representing a cart design. Table A.3 lists all GGREADA's rules for graph generation, with the exception of R_{12}, the vertex subtraction rule. That rule description follows the discussion of characteristic functions used in rule applications.

In Table A.3 each rule is described by 'before' and 'after' diagrams and vertex condition descriptions. Several characteristic functions are used in articulating the rules of the graph grammar. Each of these functions takes a vertex or vertex pair as an argument

Table A.1. GGREADA's vertex library.

Code	Name	Function code	Function applicability codes	Weight (g)	Surface area (cm^2)	Joints (No.)
$t_{1,3}$	create rolling	1	N/A	N/A	N/A	N/A
$t_{2,3}$	support load	2	N/A	N/A	N/A	N/A
$t_{1,2}$	mount 2 parallel wheels	3	1	N/A	N/A	N/A
$t_{2,2}$	mount 1 wheel	4	1	N/A	N/A	N/A
$t_{3,2}$	provide surface area	5	2	N/A	N/A	N/A
$t_{1,1}$	flat plate	N/A	3, 4, and 5	40	72.59	4
$t_{2,1}$	flanged plate	N/A	3, 4, and 5	40	56.46	4
$t_{3,1}$	5-sided plate	N/A	3, 4, and 5	10	7.26	3
$t_{4,1}$	5-sided flanged plate	N/A	3, 4, and 5	10	4.84	3
$t_{5,1}$	angle bracket	N/A	N/A	2	0	0
$t_{6,1}$	wheel	N/A	N/A	6	0	0
$t_{7,1}$	short rod	N/A	N/A	2	0	0
$t_{8,1}$	long rod	N/A	N/A	12	0	0
$t_{9,1}$	3-hole strip	N/A	3, 4, and 5	5	2.42	3
$t_{10,1}$	5-hole strip	N/A	3, 4, and 5	10	4.03	3
$t_{11,1}$	11-hole strip	N/A	3, 4, and 5	15	8.87	5
$t_{12,1}$	2-bend strip	N/A	3, 4, and 5	20	4.03	5

Table A.2. GGREADA terminal sub-graph vertex library.

Code	Sub-graph name	Sub-graph Meccano Set components	Function applicability (codes)	Weight (g)	Surface area (cm^2)
$t_{13,1}$	sg_1_wheel_1	Angle bracket, long rod, wheel, angle bracket	Mount 1 wheel (4)	12	0
$t_{14,1}$	sg_1_wheel_2	2-bend strip, long rod, wheel	Mount 1 wheel (4) Provide surface area (5)	38	4.03
$t_{15,1}$	sg_1_wheel_3	5-sided flanged-plate, short rod, wheel, angle bracket	Mount 1 wheel (4) Provide surface area (5)	20	4.84
$t_{16,1}$	sg_1_wheel_4	3-hole strip, long rod, wheel, 3-hole strip	Mount 1 wheel (4)	28	0
$t_{17,1}$	sg_2_wheel_1	Angle bracket, long rod, wheel, wheel, angle bracket	Mount 2 parallel wheels (3)	18	0
$t_{18,1}$	sg_2_wheel_2	2-bend strip, long rod, wheel, wheel	Mount 2 parallel wheels (3) Provide surface area (5)	44	4.03
$t_{19,1}$	sg_2_wheel_3	5-sided flanged-plate, short rod, wheel, wheel, angle bracket	Mount 2 parallel wheels (3) Provide surface area (5)	26	4.84
$t_{20,1}$	sg_2_wheel_4	3-hole strip, long rod, wheel, wheel, 3-hole strip	Mount 2 parallel wheels (3)	34	0

Table A.3. GGREADA's graph grammar rules.

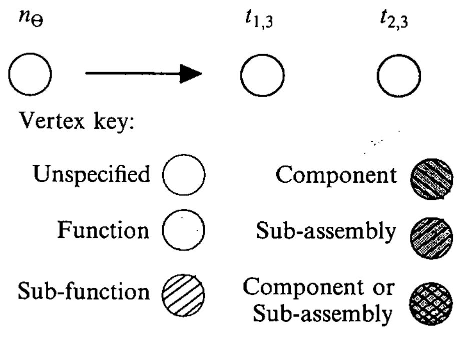
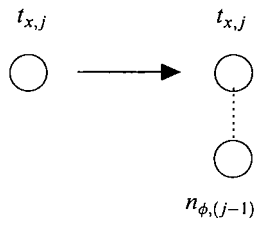
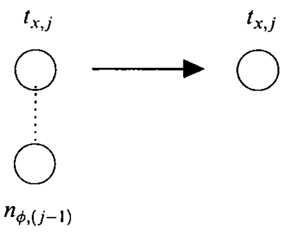
Rule	Name	Rule diagram	Description and conditions
R_1	Initial rule Transforms start symbol into function level graph for a cart. This rule is deterministic in this implementation.	 n_Θ $t_{1,3}$ $t_{2,3}$ Vertex key: Unspecified Component Function Sub-assembly Sub-function Component or Sub-assembly	After: $F_s(t_{1,3}) = 0$ $F_s(t_{1,3}) = 0$
R_2	Function status rule 1		Before: $F_s(t_{x,j}) = 0, j = 2, 3$
R_3	Function status rule 2		Before: $F_s(t_{x,j}) = 1, j = 2, 3$

Table A.3.—*(continued)*

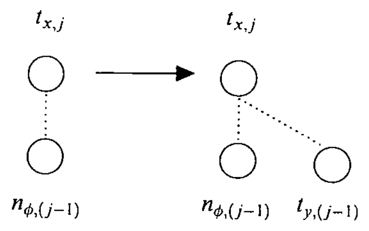
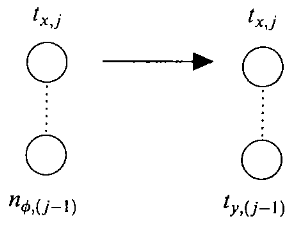
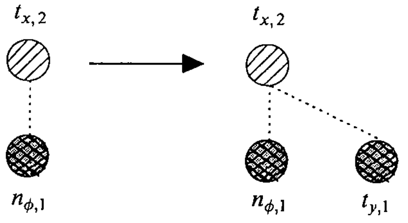
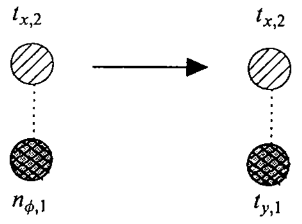
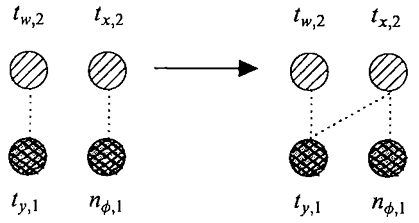
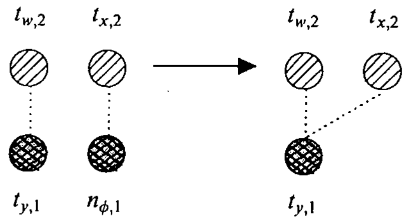
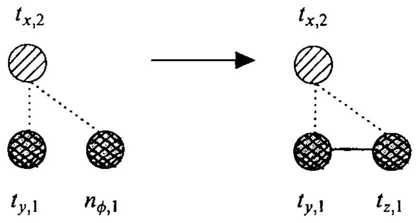
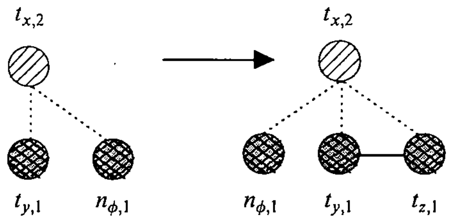
Rule	Name	Rule diagram	Description and conditions
R_4	Addition without function satisfaction	 $t_{x,j}$ $t_{x,j}$ $n_{\phi,(j-1)}$ $n_{\phi,(j-1)}$ $t_{y,(j-1)}$	Before: $F_s(t_{x,j}) = 0, j = 2, 3$ After: $F_a(t_{x,j}, t_{y,(j-1)}) = 1$ $F_s(t_{x,j}) = 0$
R_5	Addition with function satisfaction		Before: $F_s(t_{x,j}) = 0, j = 2, 3$ After: $F_a(t_{x,j}, t_{y,(j-1)}) = 1$ $F_s(t_{x,j}) = 1$
R_6	Form addition without function satisfaction	 $t_{x,2}$ $t_{x,2}$ $n_{\phi,1}$ $n_{\phi,1}$ $t_{y,1}$	Before: $\nexists t_{y,1} \in V_t(g_a)$ After: $F_a(t_{x,2}, t_{y,1}) = 1$ $F_s(t_{x,2}) = 0$
R_7	Form addition with function satisfaction		Before: $\nexists t_{y,1} \in V_t(g_a)$ After: $F_a(t_{x,2}, t_{y,1}) = 1$ $F_s(t_{x,2}) = 1$
R_8	Form assignment without function satisfaction	 $t_{w,2}$ $t_{x,2}$ $t_{w,2}$ $t_{x,2}$ $t_{y,1}$ $n_{\phi,1}$ $t_{y,1}$ $n_{\phi,1}$	Before: $F_a(t_{x,2}, t_{y,1}) = 1$ $F_{fe}(t_{y,1}, t_{x,2}) = 0$ After: $F_{fe}(t_{y,1}, t_{x,2}) = 1$ $F_s(t_{x,2}) = 0$
R_9	Form assignment with function satisfaction	 $t_{w,2}$ $t_{x,2}$ $t_{w,2}$ $t_{x,2}$ $t_{y,1}$ $n_{\phi,1}$ $t_{y,1}$	Before: $F_a(t_{x,2}, t_{y,1}) = 1$ $F_{fe}(t_{y,1}, t_{x,2}) = 0$ After: $F_{fe}(t_{y,1}, t_{x,2}) = 1$ $F_s(t_{x,2}) = 1$

Table A.3.—*(continued)*

Rule	Name	Rule diagram	Description and conditions
R_{10}	Form addition and attachment with function satisfaction		Before: $F_{\mathrm{fe}}(t_{y,1}, t_{x,2}) = 1$ After: $F_{\mathrm{a}}(t_{x,2}, t_{z,1}) = 1$ $F_{\mathrm{c}}(t_{y,1}, t_{z,1}) = 1$ $F_{\mathrm{pe}}(t_{y,1}, t_{z,1}) = 1$ $F_{\mathrm{s}}(t_{x,2}) = 1$
R_{11}	Form addition and attachment without function satisfaction		Before: $F_{\mathrm{fe}}(t_{y,1}, t_{x,2}) = 1$ After: $F_{\mathrm{a}}(t_{x,2}, t_{z,1}) = 1$ $F_{\mathrm{c}}(t_{y,1}, t_{z,1}) = 1$ $F_{\mathrm{pe}}(t_{y,1}, t_{z,1}) = 1$ $F_{\mathrm{s}}(t_{x,2}) = 0$

and returns 1 if a condition is met and 0 otherwise. The functions are defined as follows:

$F_{\mathrm{s}}(v)$, satisfaction characteristic function: returns 1 if the functionality of the vertex entity is satisfied, 0 otherwise.

$F_{\mathrm{a}}(u, v)$, applicability characteristic function: returns 1 if the behaviors of the entity of vertex v can be applied to satisfy all or part of the functionality required by the entity of vertex u, 0 otherwise.

$F_{\mathrm{c}}(u, v)$, connectivity characteristic function: returns 1 if the conditions between the entities of vertices u and v exist for making a physical connection between them. To make a connection, the number of joints required for a new form to connect to the existing graph must be available on the base piece.

$F_{\mathrm{fe}}(u, v)$, functional edge characteristic function: returns 1 if there is a functional edge between vertices u and v, 0 otherwise.

$F_{\mathrm{pe}}(u, v)$, physical edge characteristic function: returns 1 if there is a physical edge between vertices u and v, 0 otherwise.

Rule R_{12} is the vertex subtraction rule. It does not appear in Table A.3 because it is difficult to succinctly describe in a diagram. This rule can be used to alter an existing feasible design graph by removing a terminal vertex. Edges must also be removed as part of a vertex subtraction. When a vertex is removed from a graph, all edges that ended in that vertex cease to exist as part of the graph. After removing the vertex with rule R_{12}, rule R_2 would be applied to insert one or more non-terminal vertices into the graph. The design process would then continue as when building a new design.

GGREADA's subtraction rule schema is best illustrated by describing the graph before and after a terminal vertex is removed as follows.

R_{12} vertex subraction:

To remove $t_{x,j}$ from graph g_{a}, in the rule application $R_{12} : g_{\mathrm{a}} \rightarrow g_{\mathrm{b}}$,

For $j = 1$:

$V_{\mathrm{t}}(g_{\mathrm{b}}) = V_{\mathrm{t}}(g_{\mathrm{a}}) - \{t_{x,1}\} - \{t_{y,1}\}$ for all $t_{y,1}$ such that $t_{y,1}$ is a sub-graph vertex and there is no physical edge $(t_{y,1}, t_{z,1}) \in E_{\mathrm{p}}(g_{\mathrm{a}})$ with $z \neq x$.

This rule states that Level 1 sub-graph components that are physically connected only to the component vertex being removed, most also be removed.

For $j > 1$:

$V_{\mathrm{t}}(g_{\mathrm{b}}) = V_{\mathrm{t}}(g_{\mathrm{a}}) - \{t_{x,j}\} - \{t_{y,k}\}$ for all $t_{y,k}$ such that $t_{y,k}$ is vertex with $k < j$ and there is no functional edge $(t_{y,k}, t_{z,j}) \in E_{\mathrm{f}}(g_{\mathrm{a}})$ with $z \neq x$.

This rule states that vertices must have a functional edge to some other vertex in the graph to remain in the graph.