

Design-as-satisfiability: A new approach to automated synthesis

DAN BRAHA*

¹Department of Industrial Engineering, Ben-Gurion University, P.O.B. 653, Beer-Sheva 84105, Israel

(RECEIVED December 1, 2000; ACCEPTED April 29, 2001)

Abstract

This article addresses computational synthesis systems that attempt to find a structural description that matches a set of initial functional requirements and design constraints with a finite sequence of production rules. It has been previously shown by the author that it is computationally difficult to identify a sequence of production rules that can lead to a satisficing design solution. As a result, computational synthesis, particularly with large volumes of selection information, requires effective design search procedures. Many computational synthesis systems utilize transformational search strategies. However, such search strategies are inefficient due to the combinatorial nature of the problem. In this article, the problem is approached using a completely different paradigm. The new approach encodes a design search problem as a Boolean (propositional) satisfiability problem, such that from every satisfying Boolean-valued truth assignment to the corresponding Boolean expression we efficiently can derive a solution to the original synthesis problem (along with its finite sequence of production rules). A major advantage of the proposed approach is the possibility of utilizing recently developed powerful randomized search algorithms for solving Boolean satisfiability problems, which considerably outperform the most widely used satisfiability algorithms. The new design-as-satisfiability technique provides a flexible framework for stating a variety of design constraints, and also represents properly the theory behind modern constraint-based design systems.

Keywords: Conceptual Design; Constraint Satisfaction; Design Search; Knowledge-Based Design; Satisfiability; Synthesis

1. INTRODUCTION

1.1. The design synthesis problem

Design synthesis may be viewed as the transformation of an abstract functional description for a device into a structural description that satisfies the functional requirements (Pahl & Beitz, 1988; Chandrasekaran, 1990; Maher, 1990; Sriram, 1997; Sabin & Weigel, 1998). The above characteristic of design synthesis as mapping function to form is often mitigated by decomposing the synthesis task into a hierarchical one (Brown & Chandrasekaran, 1989; Chandrasekaran, 1990; Maher, 1990). Some design synthesis systems have used the decomposition model by representing much of the knowledge about the problem declaratively,

usually in the form of causal decompositions, production rules, or transformational rules (Chandrasekaran, 1990; Coyne et al, 1990; Maher, 1990; Bradley et al., 1993; Dym, 1994; Maimon & Braha, 1996; Brown, 1997; Braha & Maimon, 1998).

Rule-based (expert) systems have been applied to assist in a variety of engineering design tasks such as: design for VAX computer systems by DEC (R1—McDermott, 1982, 1993), design system for small computers (M1—Brown & Chandrasekaran, 1989), configuration of microcomputer systems (COSSACK—Frayman & Mittal, 1987), design of air cylinders (AIR-CYL—Brown & Chandrasekaran, 1985, 1989), design of elevators (VT—Marcus et al., 1988; Schreiber & Birmingham, 1996), design of facilities on construction sites (SightPlan—Tommelein et al., 1991), design of buildings (HI-RISE—Maher, 1988; CONGEN—Sriram, 1997), design of paper-feeding mechanisms of photocopiers (PRIDE—Mittal & Dym, 1986; or Koo et al., 1998), design of pneumatic systems (PNEUDES—Shin & Lee, 1998), design of VLSI circuits (VEXED—Mitchell et al., 1985), and diagnosis and selection (Brown, 1998). See also Tong and Sriram (1991) for most of these systems.

*Present address: Center for Innovation in Product Development, Massachusetts Institute of Technology, Room E60-236, 30 Memorial Drive, Cambridge, MA 02139. E-mail: Braha@mit.

Reprint requests to: Dan Braha, Department of Industrial Engineering, Ben-Gurion University, P.O.B. 653, Beer-Sheva 84105, Israel. E-mail: brahad@bgumail.bgu.ac.il

Grammatical design is another paradigm based directly on the view of design as the process of transforming an initial set of requirements into an explicit, complete specification of an object that satisfies those requirements (Brown, 1997). A grammar is a formal generative device consisting of a vocabulary of elements, a set of production rules that transform structured arrangements of the elements into new structures, and an initial structure (Gips & Stiny, 1980). Design using grammar involves recursively selecting transformation rules and applying them to a candidate structure, until a final structure that satisfies design requirements emerges (Brown, 1997). Shape grammars use symbols that are based on shapes made up of points and lines, and have been utilized to describe spatial design languages that capture architectural style (Gips & Stiny, 1972). Computational issues of implementation shape grammars have been considered by some authors (e.g., Chase, 1989; Krishnamurti, 1992). Grammatical design has also been used to describe languages for engineering design purposes. Early applications of grammars in engineering appeared in solid modeling representations (e.g., Fu et al., 1993). Other types of grammars in engineering design, which employ higher levels of description include graph grammars and attribute grammars for mechanical devices (Mullins & Rinderle, 1990), a labeled parametric shape grammar for the manufacturing process plan (Brown et al., 1994), a spatial and functional grammar for structural design (Fenves & Baker, 1987), and a bond graph grammar for mechanical systems (Hoover & Rinderle, 1989; Ulrich & Seering, 1989; Finger & Rinderle, 1990).

In the above transformational models of design, design synthesis can be viewed as a search process that attempts to find a structural description to match a set of initial functional requirements and design constraints with a finite sequence of operators or production rules included in the designer's knowledge body (Chandrasekaran, 1990; Maher, 1990; Dym, 1994; Brown, 1997; Sriram, 1997; Wielinga & Schreiber, 1997; Braha & Maimon, 1998). A process step corresponds to the transformation (transition) from one process state to another. A transformation, which describes the relation between two adjacent process states, is activated by applying one of a finite set of operators. A design process is a series of transformations. According to this model, the initial process state, that is, the abstract, functional specification of the target artifact, is known a priori. The "accepting" (satisficing) process state that provides the structural description of the artifact is to be determined. At each step, an operator transforms the given process state into a different process state. The task is to find a sequence of operators (production rules) that will lead to an accepting process state. This sequence of operators can be viewed as decomposing the initial abstract specification into a concrete structural description. The set of all process states that can be reached by applying production rules is called the *state space*. The states in the state space that are accepting constitute the *solution space*.

Depending on the initial functional requirements and design constraints, there may exist a large number of sequences of production rules that lead to structural descriptions. Only a miniscule number of structural descriptions in the solution space constitute satisficing solutions. Thus, the task of looking for the sequence of production rules that will transform the initial functional requirements into an accepting structural description of the device (as defined by the requirements and constraints) may be computationally intractable. To capture rigorously the computational complexity of a design synthesis process with the above-mentioned characteristics, the Design Process (DP) decision problem was formulated in Braha and Maimon (1998) as follows: "Is there a finite sequence of production rules that begins with the initial process state and ends with an accepting process state?" In Braha and Maimon (1998), it was shown that the DP decision problem is computationally difficult (i.e., NP-complete; see Garey & Johnson, 1979) even for restricted propositional-based knowledge representation schemes. In particular, this means that that CPU time required to solve the DP decision problem, based on known algorithms, grows exponentially with the "size" (roughly speaking, "size" means the total number of production rules, structural and functional attributes) of the problem. At this point in time, no polynomial time algorithms exist that are capable of solving NP-complete problems, and it is unlikely that polynomial time algorithms will be developed for these problems. The above complexity result concludes that design problem solving, particularly with its large volume of selection information, requires the aid of computationally efficient search algorithms (i.e., brute force is inefficient).

Various control strategies have been developed for selecting the sequence of transformations in rule-based or transformational systems, for example, inference mechanisms (Chandrasekaran, 1990; Coyne et al., 1990; Sriram, 1997); propose and backtrack or propose and revise (Wielinga & Schreiber, 1997); component-directed transformations (Hoover & Rinderle, 1989); and shape annealing (Schmidt & Cagan, 1995). However, many search algorithms have become unacceptably inefficient due to the combinatorial nature of the domain (Brown & Chandrasekaran, 1989; Schmidt & Cagan, 1995; Chakrabarti & Bligh, 1996; Wielinga & Schreiber, 1997). Consequently, design synthesis automation becomes limited; for instance, by using a restricted representation for design problems and solutions (Ulrich & Seering, 1989; Schmidt & Cagan, 1995; Chakrabarti & Bligh, 1996) or by considering a small number of possible transformations at each selection point (Brown & Chandrasekaran, 1985; Chandrasekaran, 1990; Wielinga & Schreiber, 1997). Moreover, not many prototypes demonstrated beneficial applicability to real-world scenarios (e.g., Mitchell et al., 1985; Tong & Sriram, 1991; Schreiber & Birmingham, 1996).

Several methods have been suggested to manage the complexity associated with the extensive search in the space of possible transformations: Some methods limit the represen-

tation of design problems and solutions (Ulrich & Seering, 1989). These include, for example, arbitrarily choosing the maximum number of elements to be used in a design solution (Chakrabarti & Bligh, 1996), controlling the granularity of building blocks (Lee et al., 1992), focusing on serial machines (Schmidt & Cagan, 1995) or a single input output system (SISO—Ulrich & Seering, 1989; Chakrabarti & Bligh, 1996), or using a few important parameters at a time (Lee et al., 1992). Other methods presuppose routine design tasks (Brown & Chandrasekaran, 1985, 1989; Chandrasekaran, 1990; Brown, 1998). In routine design, an extensive search in the space of possible transformations is avoided by limiting the number of possible decompositions at each selection point to one or a few (Chandrasekaran, 1990). In domains where multiple decompositions are possible, and there are no easily established heuristics to help choose among them, finding the appropriate decomposition is computationally expensive (Chandrasekaran, 1990). In this case, in most implemented transformational-based systems, humans choose from a set of alternative transformations presented by the design synthesis system (Maher, 1990; Brown, 1998). However, in some domains manual selection is no longer feasible. For example, in configuration design (which is a restricted form of design; see Wielinga & Schreiber, 1997; Brown, 1998) the increased trend toward mass customization has awakened great interest in automated synthesis (Sabin & Weigel, 1998). Another approach for avoiding the extensive search is the use of a significant amount of domain-specific knowledge and special-purpose synthesis algorithms; however, constructing and maintaining a particular application may be time consuming and expensive (Sabin & Weigel, 1998). For example, several knowledge-intensive decomposition strategies are based on a *design plan*. A design plan represents a precompiled partial solution to a design goal (Brown & Chandrasekaran, 1985, 1989; Chandrasekaran, 1990; Sabin & Weigel, 1998). A design plan is a knowledge structure that describes how a particular requirement or subproblem can be solved. This method assumes that there is localized knowledge for handling constraint violations. It presupposes that there is a functional architecture with a mapping of design functions to components (Sabin & Weigel, 1998). Knowledge-directed synthesis strategies have also been developed for grammatical design (Brown, 1997). For example, the “component-directed transformation” described in Hoover and Rinderle (1989) is a synthesis strategy for mechanical devices. These are guided by functional integration or function sharing and incidental behavior principles, and by knowledge of the available components of the domain.

1.2. The design-as-satisfiability approach

To address the combinatorial nature of rule-based or transformational design synthesis problems, we present a new, fully automated mechanism for solving a broad and inter-

esting class of design synthesis problems. We focus on conceptual (“preliminary”) design synthesis (Pahl & Beitz, 1988; Ulrich & Seering, 1989; Chakrabarti & Bligh, 1996; Brown, 1998) that incorporates a set of production rules to represent the design synthesis knowledge.

The new approach is based on transforming the problem of reaching an acceptable solution to that of finding a satisfying assignment for an expression in Boolean (propositional) logic. This problem can be given to a specialized algorithm for solving Boolean satisfiability problems (also called SAT engine), and from every satisfying Boolean-valued truth assignment to the corresponding Boolean expression we efficiently can derive a solution to the original design problem along with its finite sequence of production rules (see Fig. 1). The general methodology presented in Figure 1 is further detailed in Section 3. In order to illustrate the main ideas and *without loss of generality*, we assume in the following that the design representation scheme (e.g., of production rules) is expressible in propositional logic (i.e., the structural and functional attributes are considered *0-ary predicates*). We illustrate the approach to a conceptual design of automobiles. In Appendix A, the applicability of the new *design-as-satisfiability* approach to rule-based design knowledge is shown where the rule elements are first-order logic-based predicates. We illustrate the approach to a conceptual design of machines (see Schmidt & Cagan, 1995). This extension is contrasted with transformational approaches for machine design; such as abstraction grammars (Schmidt & Cagan, 1995), bond graph grammars (Hoover & Rinderle, 1989; Ulrich & Seering, 1989; Finger & Rinderle, 1990), compositional synthesis (Chakrabarti & Bligh, 1996), or predicate logic-based synthesis (Kannapan & Marshek, 1990).

While the proposed approach may seem roundabout, it is made attractive by the recent development of powerful SAT engines that are based on a stochastic local search technique. The approach has also proven useful in the related problem of generative planning (Kautz & Selman, 1996;

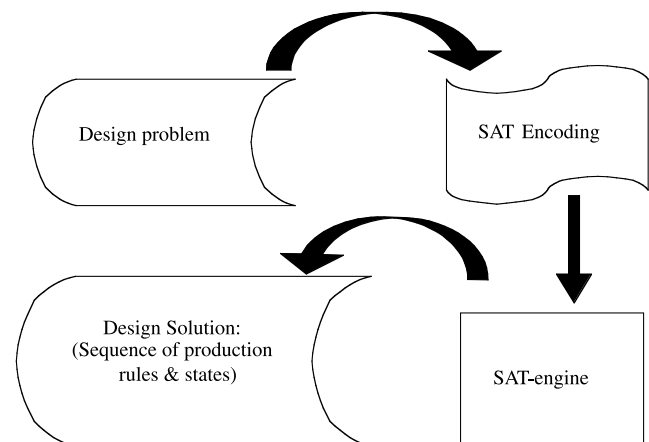


Fig. 1. The SAT encoding methodology for design search.

Kautz et al., 1996; Ernst et al., 1997). These randomized search algorithms can be used for solving nontrivial satisfiability problems that are an order of magnitude larger than can be solved by the most widely used satisfiability algorithms (e.g., the Davis–Putnam procedure). The proposed approach provides a more flexible framework for stating different kinds of design plan constraints (e.g., decoupling of coupled design), and also more accurately reflects the theory behind modern constraint-based design systems.

The article is organized as follows. To provide a basis for contrasting the new method, a design synthesis example that uses an inference mechanism similar to those used in some rule-based (expert) systems is presented in Section 2. In Section 3, the new *design-as-satisfiability* approach, which is based on finding an appropriate SAT encoding for the design problem, is introduced. Conclusions are drawn in Section 4. Several extensions of the new approach are presented in Appendix A.

2. A RULE-BASED DESIGN EXAMPLE

In this section, we present a simple rule-based design example in order to plainly illustrate the type of design problems considered in this article. The design example will be solved using a traditional rule-based design system. In Section 3, the same example will be solved using the new *design-as-satisfiability* approach in order to highlight the disparity in methodologies.

Rule-based design systems mainly contain domain-specific knowledge (facts and/or rules) and employ a separate inference procedure to manipulate this knowledge (e.g., Coyne et al., 1990; Huang & Brandon, 1993; Dym, 1994; Sriram, 1997). The main components of a rule-based design system are: (1) the knowledge base, which contains all the information associated with the domain in which the system is to operate. This information may be facts, rules, as well as rules of thumb and heuristics. (2) The working memory (also known as context, or short-term memory), which contains all the information about the problem currently being solved. Its content changes dynamically and includes information that defines the parameters of the specific problem and information derived by the system at any stage of the solution process. (3) The inference mechanism (also known as the inference engine, control mechanism, or reasoning mechanism), which controls the reasoning process of the system. The inference mechanism uses the knowledge base to modify and expand the context in order to solve a specific design problem.¹

Since a problem space might be enormously large, the inference engine uses heuristics (“rules of thumb”) to effi-

ciently control it, and by that provide a right solution *quickly*. The global control of a rule-based design system is either forward (data driven) or backward chaining (goal driven; Coyne et al., 1990). Using forward chaining, we start from the antecedent of the rule² (known facts, “if” clause) towards the consequent (“then” clause) that needs to be achieved. This kind of deductive inference is useful in analysis. The converse of this strategy is called backward chaining, which is a kind of abduction. Synthesis, which is the task of finding a structure given a functional requirement, is likely to be abductive rather than deductive (Coyne et al., 1990). Here, the strategy is to begin with the consequent (the “goal state” or the “functional requirements”) we want to be satisfied. The conditions for the accomplishment of the functional requirements are identified and these become subgoals. The search process then consists of recursively selecting and applying transformation rules to a candidate subgoal. The search process terminates whenever functional requirements, which correspond to known structural attributes, are identified. If incompatibilities (e.g., violated constraints) happen in later design phases, then backtracking is exercised and the inference engine generates another set of subgoals. To mechanize an abductive inference engine, it is important to generate “effective” subgoals in a controlled manner as well as employing efficient backtracking methods to deal with the enormous search space of possible goals.

In the following, the problem of designing an automobile using a rule-based system is considered. The automobile example is presented as a representative design domain where the knowledge representation scheme (e.g., of production rules) is expressible in propositional logic (i.e., rule elements to be 0-ary predicates or propositions). In Section 3.1, the new *design-as-satisfiability* approach is illustrated using the automobile design example. In Appendix A, an example of a serial machine design³ is presented where the rule elements are first-order logic-based predicates; and it is shown how to apply the *design-as-satisfiability* approach to this type of problems.

2.1. Automobile design example

As an example, consider the problem of designing an automobile using a rule-based system. The automobile is a self-propelled, four-wheeled, steerable vehicle for transporting people on land. All passenger cars, trucks, and buses have certain things in common: (1) the power plant, or engine; (2) the chassis, which supports the engine and wheels and includes the frame and the steering and brake systems; (3) the power train, which transmits the power from the engine

¹Additional components such as a user interface and an explanation facility are required in order to facilitate the use of the rule-based expert system. Both learning and knowledge acquisition tools are also desirable in order to ease the development of the knowledge base.

²Here, in any formula of the form $A \rightarrow B$, A is referred to as the *antecedent* and B as the *consequent*.

³A similar problem is formulated and solved differently in Schmidt and Cagan (1995), who use an *abstraction grammar* (a production system) for the representation and generation of function and serial form layouts.

to the car wheels; and (4) the body. Technical and operational details related to the design of the main parts of automobiles and their components are provided in Braha and Maimon (1998). The compiled rules below rely very heavily on this domain-specific knowledge.

The various possible facts (facts appear in a typewriter-like font) are described below (in a particular synthesis problem only *part*, and certainly not contradictory facts, are considered).

Facts:

- (f_1) the car is used for family driving
- (f_2) the external conditions are good
- (f_3) the maximum allowed speed is 160 Km/h
- (f_4) the car is used for urban driving

- (f_5) the maximum allowed speed is 200 Km/h
- (f_6) The car is used for sport driving
- (f_7) the car is used for executive driving
- (f_8) the car is used for material transportation
- (f_9) the car is used as taxicab

The structural attributes (design description properties) that specify the configuration of actual cars as well as the functional attributes that are manifested by these structural attributes are presented in Table 1.

A small sample of the domain-specific knowledge relevant to the car design domain is expressed in terms of the production rules presented in Appendix B. The set of rules is held in a *rule memory*.

Table 1. Structural and functional attributes for the automobile design example

| Structural Attributes | |
|--|---|
| (s_1) 4-wheel drive | (s_{23}) high transmission ratio |
| (s_2) 4-wheel steering | (s_{24}) horn |
| (s_3) 6–8 cylinders | (s_{25}) hydraulic disk brakes |
| (s_4) absorbent front end | (s_{26}) large pistons & cylinders |
| (s_5) air bag | (s_{27}) light weight |
| (s_6) air-cooled engine | (s_{28}) liquid cooling system |
| (s_7) air deflector | (s_{29}) low & small structure |
| (s_8) an engine that deflects down | (s_{30}) muffler |
| (s_9) anti-lock braking system (ABS) | (s_{31}) power brakes |
| (s_{10}) automatic belts | (s_{32}) powerful starter |
| (s_{11}) catalytic converter | (s_{33}) radial tire |
| (s_{12}) deep thread patterns | (s_{34}) richer mixture fuel |
| (s_{13}) disconnecting fan system | (s_{35}) rigid passenger compartment |
| (s_{14}) drum brakes | (s_{36}) stabilizers in the front |
| (s_{15}) electric powered | (s_{37}) suspension system |
| (s_{16}) electronic ignition | (s_{38}) tubeless tire |
| (s_{17}) extra differential | (s_{39}) windshield defroster |
| (s_{18}) extra strong door | (s_{40}) windshield washer & wiper |
| (s_{19}) extra strong roof | (s_{41}) tire with 205 width symbol |
| (s_{20}) fog lights | (s_{42}) tire with 155 width symbol |
| (s_{21}) fuel injection | (s_{43}) tire with 60 aspect ratio symbol |
| (s_{22}) high ground clearance | (s_{44}) tire with U speed symbol |
| Functional Attributes | |
| (r_1) AERODYNAMIC DESIGN | (r_{16}) PASSABLE IN DIFFICULT TERRAIN |
| (r_2) DIESEL ENGINE | (r_{17}) RELIABLE TIRE |
| (r_3) CREATES MINIMAL POLLUTION | (r_{18}) SAFE CAR |
| (r_4) EASY PARKING | (r_{19}) SAFE IN ACCIDENTS |
| (r_5) EFFICIENT ENGINE | (r_{20}) SAFE IN BAD WEATHER |
| (r_6) ECONOMICAL | (r_{21}) SAFE IN FLIPPING OVER |
| (r_7) RELIABLE BRAKES | (r_{22}) SAFE IN HEAD-ON COLLISIONS |
| (r_8) HEAVY CAR | (r_{23}) SAFE IN HIGH DRIVING SPEED |
| (r_9) HIGH POWER OUTPUT | (r_{24}) SAFE IN OFF-HIGHWAY ROAD |
| (r_{10}) HIGH DRIVING SPEED | (r_{25}) SAFE IN POOR EXTERNAL CONDITIONS |
| (r_{11}) HIGH VOLUME OF THE COMBUSTION CHAMBER | (r_{26}) SAFE IN POOR VISIBILITY |
| (r_{12}) LOW FUEL CONSUMPTION | (r_{27}) SAFE IN SIDE COLLISIONS |
| (r_{13}) LOW MAINTENANCE COSTS | (r_{28}) SMALL CAR |
| (r_{14}) MECHANICALLY DEPENDABLE and DURABLE | (r_{29}) SMALL ENGINE |
| (r_{15}) OFF-HIGHWAY TIRE | (r_{30}) HIGH-POWERED ENGINE |

2.2. Car synthesis trace

Assume that the designer is faced with the problem of designing a car description that is able to achieve the following functional attributes as requirements:

1. The car CREATES MINIMAL POLLUTION (r_3)
2. The car is capable of HIGH DRIVING SPEED (r_{10});
3. The car has LOW FUEL CONSUMPTION (r_{12});
4. The car is SAFE (r_{18}).

The following facts are assumed: (1) the car is used for family driving (f_1); (2) the external conditions are good (f_2); (3) the maximum allowed speed is 160 Km/h (f_3); and (4) the car is used for urban driving (f_4).

The following *constraints*⁴ are further considered: (1) the structural attributes “tire with 205 width symbol (s_{41})” and “tire with 155 width symbol (s_{42})” *cannot* be included together in a design description; and (2) the structural attribute “the car is electric-powered (s_{15})” *cannot* be included in a design description if the functional attribute “the car has DIESEL ENGINE (r_2)” is satisfied.

The rule-based design system needs to search through the problem space for a pathway from the initial requirements to some state of the car structural description such that the requirements r_{18} , r_{10} , r_{12} , and r_3 are achieved, while adhering to the compatibility constraints.

In attempting to achieve the initial requirements, a backward chaining inference is served as the problem-solving strategy by the rule-based design system (Coyne et al., 1990; Dym, 1994; Sriram, 1997). Other control issues are addressed as follows: (1) the inference engine identifies (by matching) several production rules for which their *consequent* parts satisfy the state of the working memory. In this case, the inference engine selects a rule from the conflict set to fire. (2) The order in which the subrequirements are processed is according to a depth-first control strategy (e.g., if the functional attributes r_1 **and** r_2 are pending, expand r_1 before r_2); finally, (3) if any of the above constraints is violated during the search (a “failure;” see Brown & Chandrasekaran, 1989), *dependency-directed* backtracking⁵ (Brown & Chandrasekaran, 1989) is taken and an alternative rule is chosen from the list of available finite choices.

A *working memory* holds or represents the “current” state of the process, which is in the example a *conjunction* of structural and functional attributes (if a functional attribute appears in a process state, it means that it remains to be

⁴Constraints in *discrete* domains can be expressed as compatibility relations between attributes, stating that certain combinations are allowed or not (Sriram, 1997).

⁵Dependency-directed backtracking provides a way of taking into account the information about which pieces of knowledge contribute to the failure. This information is used in the decision of how far to backtrack (see Brown & Chandrasekaran, 1989). Dependency-directed backtracking is also related to “belief revision” (Brown & Chandrasekaran, 1989).

satisfied). Table 2⁶ shows the trace of “process states” generated in the course of searching for a solution to the automobile design problem. At each step of the search, the consequent parts of the production rules (listed above) are matched against the current unsatisfied functional attributes (given in the second column of Table 2) and known facts. Since many production rules may match the current state, the preferred rule is the one that matches the first leftmost unsatisfied functional attribute (a depth-first control strategy).

3. A SAT ENCODING FOR DESIGN SEARCH

The main objective of the forthcoming sections is to present a new computational search approach for solving design synthesis problems that incorporate a set of production rules for the design knowledge representation.

The proposed method is to recast the design problem as a *Boolean satisfiability* problem, which can be solved using a SAT engine (see Fig. 1). The existence of powerful new algorithms for solving Boolean satisfiability problems makes this technique even more appealing (Gu, 1992; Levesque et al., 1992; Trick & Johnson, 1993; Selman et al., 1996, 1997).

In Boolean satisfiability problems, we must find a truth assignment for some given set of atomic propositions that will make a given set of expressions evaluated to TRUE. Such an assignment is called a *satisfying* assignment. For example, consider the following Boolean expression in *conjunctive normal form*,⁷ which is the form used by most satisfiability-testing programs:

$$(p_1 \vee \neg p_2 \vee p_3) \wedge (p_1 \vee p_2 \vee \neg p_4) \wedge p_2$$

One satisfying assignment for it would assign TRUE to p_1 and p_2 and FALSE to all other variables.

The problem of finding a satisfying assignment for a set of Boolean expressions is known to be NP-Complete (Garey & Johnson, 1979). Hence it is unlikely that algorithms exist that are guaranteed to solve it in polynomial time. However, in practice there is a class of algorithms that have been shown to solve extremely large and difficult satisfiability problems in reasonable time (Gu, 1992; Levesque et al., 1992; Selman et al., 1996, 1997). These algorithms (e.g., GSAT and Walksat; Selman et al., 1996), are based on stochastic local search methods that perform “noisy” greedy searches, that is, where each search step usually moves from the current candidate solution to the best neighboring solution (according to some neighborhood relation), but can occasionally make random moves that are not locally optimal. These randomized greedy algorithms were used to solve empirically hard random formulas as well as encoding of

⁶The abduction process in a rule-based system can also be illustrated using a derivation graph (Coyne et al., 1990).

⁷A Boolean expression is in *conjunctive normal form* if $\phi = C_1 \wedge C_2 \wedge \dots \wedge C_m$, and each of the C_j s is the disjunction of one or more literals. The C_j s are called the *clauses* of the expression in conjunctive normal form.

Table 2. Design search algorithm applied to the automobile design problem^a

| Process Step | Structural Attributes + Unsatisfied Functional Attributes | Satisfied Functional Attributes | Candidate Rules | Selected Rule |
|--------------|--|--|--|---|
| 1 | r_{18} and r_{10} and r_{12} and r_3 | \emptyset | Rule 4 | Rule 4 is fired to decompose r_{18} : “the car is SAFE” |
| 2 | r_{19} and r_{10} and r_{12} and r_3 | r_{18} | Rule 6 | Rule 6 is fired to decompose r_{19} : “the car is SAFE IN ACCIDENTS” |
| 3 | r_{22} and r_{27} and r_{21} and s_{10} and r_{10} and r_{12} and r_3 | r_{19} and r_{18} | Rule 11 | Rule 11 is fired to decompose r_{22} : “the car is SAFE IN HEAD-ON COLLISIONS” |
| 4 | s_4 and s_5 and s_8 and r_{27} and r_{21} and s_{10} and r_{10} and r_{12} and r_3 | r_{22} and r_{19} and r_{18} | Rule 12 | Rule 12 is fired to decompose r_{27} : “the car is SAFE IN SIDE COLLISIONS” |
| | | | • | |
| | | | • | |
| | | | • | |
| 11 | s_4 and s_5 and s_8 and s_{18} and s_{19} and s_{35} and s_{10} and s_{29} and s_7 and s_{26} and s_3 and s_{32} and s_{28} and s_{34} and s_{30} and s_{23} and s_{41} and s_{43} and s_{44} and r_{12} and r_3 | r_2 and r_{11} and r_{30} and r_1 and r_{10} and r_{21} and r_{27} and r_{22} and r_{19} and r_{18} | Rule 21, rule 38 | Rule 38 is used to decompose r_{12} : “the car has LOW FUEL CONSUMPTION” |
| 12 | s_4 and s_5 and s_8 and s_{18} and s_{19} and s_{35} and s_{10} and s_{29} and s_7 and s_{26} and s_3 and s_{32} and s_{28} and s_{34} and s_{30} and s_{23} and s_{41} and s_{43} and s_{44} and r_5 and s_{13} and s_{42} and r_3 | r_{12} and r_2 and r_{11} and r_{30} and r_1 and r_{10} and r_{21} and r_{27} and r_{22} and r_{19} and r_{18} | — | backtracking The structural attributes s_{41} (“tire with 205 width symbol”) and s_{42} (“tire with 155 width symbol”) <i>cannot</i> be included together in a design description |
| 13 | s_4 and s_5 and s_8 and s_{18} and s_{19} and s_{35} and s_{10} and s_{29} and s_7 and s_{26} and s_3 and s_{32} and s_{28} and s_{34} and s_{30} and s_{23} and s_{41} and s_{43} and s_{44} and r_{12} and r_3 | r_2 and r_{11} and r_{30} and r_1 and r_{10} and r_{21} and r_{27} and r_{22} and r_{19} and r_{18} | rule 21, rule 38 | Rule 21 is used to decompose r_{12} : “the car has LOW FUEL CONSUMPTION” |
| | | | • | |
| | | | • | |
| | | | • | |
| 18 | s_4 and s_5 and s_8 and s_{18} and s_{19} and s_{35} and s_{10} and s_{29} and s_7 and s_{26} and s_3 and s_{32} and s_{28} and s_{34} and s_{30} and s_{23} and s_{41} and s_{43} and s_{44} and s_{16} and s_{21} and s_{13} and s_{27} and s_{11} | r_3 and r_5 and r_{12} and r_2 and r_{11} and r_{30} and r_1 and r_{10} and r_{21} and r_{27} and r_{22} and r_{19} and r_{18} | STOP consistent solution is obtained | |

Complete table may be found in Braha and Maimon (1998)

hard graph-coloring problems (as well as other problems in AI). Such methods are incomplete—that is, they are not guaranteed to find a solution if one exists. However, in practice, they are very efficient.⁸

One class of problems where the strategy of encoding problems as SAT instances has been very fruitful is in *generative planning* (Kautz & Selman, 1996; Kautz et al., 1996). In generative planning, an initial state of a system, a set of operators or actions for transforming the state of the system, and a set of goal states are given. The task is to come up with a sequence of operators that will transform a system from the initial state to any one of the goal states. It is

⁸There has been considerable progress in systematic solution algorithms for SAT problems as well.

immediately apparent that this problem is very much like standard state-space search problems, such as the design search problem. However, in state-space search, we are typically interested in discovering a final, unknown *state* that meets a certain criteria, whereas in the planning problem we are interested in a *path* to a known set of final states. It is the research objective to show how methods developed for transforming generative planning problems to SAT problems in artificial intelligence can be used and modified to transform engineering design problems to SAT problems, and to demonstrate that they can be applied to nontrivial “real-world” design problems (such as automobile and aeronautics design).

3.1. Detailed description of the methodology

There are a number of methods for encoding planning problems as SAT and we will suggest a method by which any of these encoding techniques can be applied to the design search problem. However, to make the presentation of this article clearer, we will concentrate on one particular plan encoding scheme (outlined in Braha & Brafman, 1998), known as the *linear encoding*. To describe an encoding, we have to specify three things: the set of propositions used, the set of expressions (or “axioms”) to be generated, and how a solution to the original problem is obtained from the solution to the encoded SAT problem.

The idea behind the linear encoding is the following: Let n be the bound on the number of plan steps.⁹ The set of propositions is composed of two classes, where every proposition in the first class is associated with the execution of each action (operator) at each time point, and every proposition in the second class is associated with the execution of each attribute at each time point. Intuitively, we would like to obtain a truth assignment that reflects a valid solution, that is, one where the proposition associated with action A at time¹⁰ t has the value TRUE if action A is the action that should be executed at time t , and where the proposition associated with the attribute P at time t has the value TRUE if, at the state reached after t actions are performed, P holds. For this to be the case, we must specify a set of axioms such that any satisfying truth assignment for these axioms will have the above properties. The linear encoding achieves this by including the following set of axioms:

1. An initial state axiom specifying that at time 0 all (and only) attributes characterizing the initial state hold.
2. A goal state axiom that specifies that at the last time point, n , all goal attributes hold.
3. Axioms ensuring that if a particular action is taken at time t , then the attributes at time t and at time $t + 1$ are related appropriately, that is, those propositions holding at state $t + 1$ correspond to the state obtained when this action is executed at the state at time t .
4. Axioms ensuring that only one action is executed at each time point.

After generating a set of Boolean expressions that correspond to the above set of axioms, a SAT engine can be used to find a satisfying truth assignment. Given such an assignment, we can deduce the sequence of actions of the plan by seeing which action proposition holds (i.e. has the value TRUE) at each time point.

Since rule-based design searching¹¹ is so closely related to planning, it seems likely that any method of encoding planning problems as SAT problems could be used to encode rule-based design search problems into SAT problems. The only caveat is that one must be able to describe the criteria a solution process state must satisfy using propositional logic. In a class of planning problems formulated using the STRIPS representation language (Fikes & Nilsson, 1971), goal states are described via a conjunction of properties. Hence, we can encode them using a conjunction of propositions. Solution criteria for general design search problems will not necessarily have this property. In fact, at a first glance, it is not obvious that the criteria for a solution state of a design problem can be formulated this way: It requires a state in which all attributes are structural. Fortunately, it is easy to see that we can recast this requirement by stating that there will be no functional attributes in the final process state, that is, the final state axiom will assert that all state propositions corresponding to functional attributes at the final state must be false. Finally, in many design problems, there are additional constraints that must be satisfied (e.g., a 205-width-symbol tire and a 155-width-symbol tire *cannot* be included together in a design description), and these can be ensured by adding such constraints to the final state description (assuming they are expressible in propositional language).

As with the planning problems, each expression we produce corresponds to the applications of a bounded number of production rules. In the design search problem, we are interested in a final artifact’s description, which is known *a priori* to be bounded by the size of the set of structural attributes. Hence, one could object to a restriction on the number of production rules used (i.e., to the length of

⁹In the SAT-based approach, the maximal length of a plan is fixed at solving time. If the designing length is not known in advance, it is straightforward to perform both linear and binary search on designing lengths, to find the smallest for which a solution is found (Kantz & Selman, 1996). For example, if the optimal designing length is 20, the search would proceed through designing of length 2, 4, 8, 16, 32 (plan found), 24 (plan found), and finally 20. The “minimal plan” characteristic of the SAT-based approach addresses one of the plan’s desirable qualities mentioned in Brown and Chandrasekaran (1989), which is to choose the shorter plan if all else is equal.

¹⁰Actions are performed in sequence. Here t is a time-index parameter (ranging over the numbers 1, 2, ...) that is added to each action or attribute, to indicate the state at which the action begins or the attribute holds. For a problem bounded by n , the actions are indexed by 0 through $n - 1$, and the attributes by 0 through n .

¹¹Again, we focus on preliminary design that incorporates a rule-based representation of design knowledge.

the path to the solution state). But if the solution requires the application of an exponential number of production rules to be generated, it is unlikely that any method can solve it without using exponential time. However, under reasonable¹² restrictions on the form of the production rules, it is easy to see that if a solution exists then there exists a solution path that is polynomial in the number and size of the production rules.

We now show how these ideas are applied to the automobile design example presented in Section 2.

EXAMPLE 1. We shall use the proposition $r_i(t)$ [or $s_i(t)$] to denote that the functional attribute r_i (or the structural attribute s_i) holds at the state occurring at time t . The notation, for example, $r_i(t)$, is intended to be interpreted as a **name** and not as a **function** of t . We shall need one such proposition for every one of the 44 structural and 30 functional attributes at every time point $0 \leq t \leq n$. Similarly, $p_i(t)$ corresponds to the application of production rule i at time t . We shall need one such proposition for every one of the 38 production rules and for every time point $0 \leq t < n$ (we cannot apply a rule at time n , as this is the final state). In addition, propositions, $p_0(t)$, corresponding to a dummy “null” production rule are also included. These propositions handle the case where the number of process steps is actually shorter than n . Finally, $f_i(0)$ is used to denote the proposition associated with the fact f_i at time 0.¹³ We shall need the following axioms.

INITIAL STATE AXIOM. The initial state must hold at time 0:

$$\overbrace{f_1(0) \wedge f_2(0) \wedge f_3(0) \wedge f_4(0)}^{\text{known facts}} \wedge \neg f_5(0) \wedge \dots \wedge \neg f_9(0)$$

$$\overbrace{\wedge r_3(0) \wedge r_{10}(0) \wedge r_{12}(0) \wedge r_{18}(0)}^{\text{initial requirements}}$$

$$\wedge \neg r_4(0) \wedge \dots \wedge \neg r_{30}(0) \wedge \neg s_1(0) \wedge \dots \wedge \neg s_{44}(0) \quad \blacksquare$$

GOAL STATE AXIOM. A final state must hold at time n :

$$\neg r_1(n) \wedge \neg r_2(n) \wedge \dots \wedge \neg r_{30}(n) \quad \blacksquare$$

SINGLE-RULE AXIOMS. At least one production rule is applied at each time point. Hence for every $0 \leq t < n$:

$$p_0(t) \vee p_1(t) \vee p_2(t) \dots \vee p_{38}(t) \quad \blacksquare$$

No more than one production rule is applied at each time point. Hence for every $0 \leq t < n$ and for every $0 \leq i \leq 38$:

$$p_i(t) \rightarrow \left(\bigwedge_{j \neq i} \neg p_j(t) \right) \quad \blacksquare$$

PRECONDITIONS AND EFFECT AXIOMS: If a production rule is applied at time t then the states at time t and $t + 1$ are appropriate for the rule application. This implies two things. First, the consequent of the rule holds at time t (which corresponds to the functional attribute being decomposed); the antecedent of the rule, and the negation of the consequent hold at time $t + 1$ (which means that the functional attribute has been replaced by the attributes appearing at the antecedent). As an example of this class of axioms consider rule 1 in the automobile rule-base. The antecedent contains $s_1, s_{17}, s_{22}, s_{27}, r_{15}$, while the consequent is r_{16} . Hence, for every $0 \leq t < n$ we add:

$$p_1(t) \rightarrow r_{16}(t) \wedge \neg r_{16}(t+1) \wedge s_1(t+1) \wedge s_{17}(t+1) \\ \wedge s_{22}(t+1) \wedge r_{15}(t+1) \quad \blacksquare$$

FRAME AXIOMS: If the truth value of an attribute changes from true to false, then one of the production rules that includes the attribute in its consequent part must have occurred. Similarly, if the truth value of an attribute changes from false to true, then one of the production rules that includes the attribute in its antecedent part must have occurred. If those production rules don't occur, then by modus tollens the truth value of the attribute must continue through an existing production rule. For example, for every $0 \leq t < n$ we must add the following frame axiom, which says which production rules could have caused $r_3(t)$'s truth value to change:

$$r_3(t) \wedge \neg r_3(t+1) \rightarrow p_{35}(t) \vee p_{36}(t) \quad \blacksquare$$

Similar frame axioms must appear for every other attribute.

COMPATIBILITY CONSTRAINT AXIOMS: To illustrate the first constraint, that is, both structural attributes s_{41} and s_{42} cannot be included together in a physically realizable design, the following must hold at time n :

$$\neg s_{41}(n) \vee \neg s_{42}(n) \quad \blacksquare$$

Constructing and Solving the Complete Encoded Boolean Expression. Using some basic useful properties of Boolean connectives, the above set of axioms (i.e. their conjunction) can be rewritten (in polynomial computation time) into an equivalent Boolean expression in *conjunctive normal form* (which is the form used by most satisfiability-testing programs). For example, the expression, $p_1(t) \rightarrow r_{16}(t) \wedge \neg r_{16}(t+1) \wedge s_1(t+1) \wedge s_{17}(t+1) \wedge s_{22}(t+1) \wedge r_{15}(t+1)$ is rewritten as the following equivalent expression in conjunctive normal form:

¹²Here “reasonable” expresses the fact that the rules graphically correspond to an acyclic directed graph.

¹³We need one proposition per fact since we assume that facts do not change during the process (i.e., facts are determined at time 0).

$$\begin{aligned}
 &(\neg p_1(t) \vee r_{16}(t)) \wedge (\neg p_1(t) \vee \neg r_{16}(t+1)) \wedge (\neg p_1(t) \vee s_1(t+1)) \\
 &\wedge (\neg p_1(t) \vee s_{17}(t+1)) \wedge (\neg p_1(t) \vee s_{22}(t+1)) \\
 &\wedge (\neg p_1(t) \vee r_{15}(t+1))
 \end{aligned}$$

The complete encoded Boolean expression, **representing all the axioms**, is inputted to a SAT engine. If the engine outputs a satisfying Boolean-valued truth assignment, we can extract the design solution from it by collecting the set of propositions of the form $s_i(n)$ to which it assigns the value TRUE. Similarly, we can construct the finite sequence of production rules that lead to this design solution; and thus facilitate the use of the SAT-based approach by having an explanation facility. The overall *design-as-satisfiability* methodology is shown in Figure 2. The correctness of the above encoding can be shown by induction on the time index t : that is, the transformed Boolean expression is satisfiable *if and only if* there is a finite sequence of at most n production rules that begins with the initial process state and ends with an accepting process state.

From this example, it is evident that we can encode any design problem, which is expressible in propositional language or the STRIPS representation language (Fikes & Nilsson, 1971; see also Appendix A), using the same class of axioms: initial state axiom, goal state axiom, single rule axioms, preconditions and effect axioms, frame axioms, and compatibility constraints axioms. Furthermore, we see that the size of this encoding is polynomial in the size of the original problem (a more rigorous analysis is provided in Section 3.2).

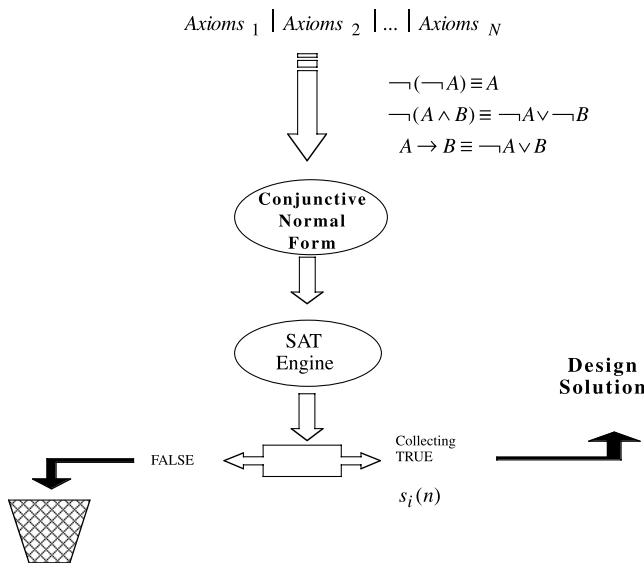


Fig. 2. Design as satisfiability: Transforming the design axioms to a Boolean expression, solving the SAT problem, and translating the satisfying truth assignment to a design solution.

3.2. The size complexity of the encoded Boolean expression

The “size complexity” of a Boolean expression is measured in terms of the total number of variables and literals it contains. A key issue in the proposed approach is the development of a practical reduction of design problems to SAT, which is measured in terms of the size complexity of the encoded Boolean expression. In spite of the fact that not all SAT problems of a given size are equally difficult,¹⁴ experimental results show (Levesque et al., 1992; Kautz & Selman, 1996; Selman et al., 1996) that formulas containing around 2,000 variables could be solved by both systematic and stochastic search in a few seconds. The limits of the systematic algorithm “tableau” were reached at 2,800 variables and 6 hours of running time (Kautz & Selman, 1996). The stochastic algorithm “Walksat” (see Selman et al., 1996) was reported to solve problems containing 10,000–100,000 variables. These stochastic search algorithms were often evaluated for the worst possible instances [i.e., on “hard” random formulas (Selman et al., 1996; Cook & Mitchell, 1997)]. In practice, larger problems (in terms of total number of variables) are expected to be considerably easier to solve than suggested by those worst case estimates. For example, in routine design, the number of possible decompositions at each selection point is limited to one or a few (Brown & Chandrasekaran, 1989, p. 112; Chandrasekaran, 1990, p. 64). In terms of the frame axioms, it is translated into a small number of rules that can account for a change in the truth value of an attribute. This results in Boolean expressions that are easily solved, although the number of variables may be high. It therefore appears that in the case of routine design the worst-case results may be overly pessimistic. In addition, in some transformational systems the number of rules that could lead to a design solution is small [roughly speaking, this number is related to the “depth” of the hierarchical decomposition (Brown & Chandrasekaran, 1989, p. 107; Schmidt & Cagan, 1995, p. 109; Chakrabarti & Bligh, 1996, p. 318)]. Again, this results in Boolean expressions that are easily solved.

The size of the encoded Boolean expression is affected by the various characteristics of the original design problem. In this section, we present an analysis of the size of the encoded Boolean expression as a function of various properties of the design problem instance; such as the number of structural and functional attributes, and the number of production rules. This type of analysis allows the user to select an efficient SAT engine based on the gross statistical properties (i.e., total number of production rules and attributes) of the statement of a given design problem. This kind of analysis is also useful when comparing different encoding

¹⁴Statistical properties of randomly generated “hard” SAT problems (as observed by numerical simulations; see Selman et al., 1996), show that *both* the number of variables and literals affect the efficiency of the solution procedure.

techniques (other than the linear encoding scheme considered in this article) for the given design problem.

In the following analysis, it is assumed that the design problem is expressible in propositional language (as in the automobile design example). The linear encoding scheme is critically dependent on the number of production rules $|R|$, number of structural and functional attributes $|A|$, and process length bound n . Based on these design features, we observe that the number of variables used in the encoding is¹⁵ $O(n|R| + n|A|)$. The rate of growth of the size of the *conjunctive normal form* expression (i.e., number of literal occurrences) is captured by the Single-Rule and Frame axioms. Consequently, the size of the transformed expression in *conjunctive normal form* is $O(n|R|^2 + n|A||R|)$.

To evaluate the performance of the SAT encoding approach, six design problems have been examined. The characteristics of the underlying design problems (e.g., automobile and forklift design with 150 attributes and production rules) have been published in Braha (2000). The results demonstrate the effectiveness of the SAT encoding approach in terms of yielding the right answer quickly (a few seconds of CPU time). Although the computational efficiency has been the primary motivation for investigating design as satisfiability, the formalization of rule-based design synthesis as satisfiability has also been effective in terms of providing a flexible framework for stating different design constraints (see Sec. 4.2).

4. SUMMARY AND DISCUSSION

4.1. Summary

It has been recognized that although the consideration in developing some rule-based synthesis systems is defining the set of productions for representing design knowledge, the difficult question for the practical use of rule-based synthesis systems is how rule execution is controlled (Maher, 1990; Brown, 1997). The computational complexity results presented in Braha and Maimon (1998) suggest that we cannot hope for automatic means to search through a space of designs that are guaranteed to work well (in the sense of providing the right answer *quickly*) for all inputs. In this article, we have proposed a new alternative, fully automated approach for solving design synthesis problems that incorporate a set of production rules for the design knowledge representation. The SAT encoding method is based on the idea of producing a Boolean expression that “represents” the original problem in the sense that from every satisfying truth assignment to this expression we can (efficiently) derive a solution to the original problem. By applying the method to several design problems, the SAT encoding approach has been found to be appealing in terms of yield-

ing the right answer quickly, and providing a flexible framework for stating different design constraints. The SAT encoding has also provided a new perspective on representational issues in design; thus, the approach is interesting from a purely representational point of view.

4.2. Discussion

Although the computational efficiency has been the primary motivation for investigating design as satisfiability, the formalization of rule-based design synthesis as satisfiability has a number of additional attractive properties.

The design-as-satisfiability technique provides a flexible framework for stating different kinds of design plan constraints; for example, by stating arbitrarily “facts” about the initial and goal states of the design process, or by stating arbitrary constraints in any intermediate state of the synthesis process. To illustrate, consider a domain-dependent control strategy that involves *prioritizing* the order in which the functional requirements (“goals”) should be processed (e.g., Coyne et al., 1990). For example, let us assume that the requirement (r_3) “the car creates minimal pollution” should be processed *before* the goal (r_{10}) “the car is capable of high speed.” If we want to insure that the above “priority” control strategy is applied, we simply add the following axioms to the SAT problem specification: for every $0 \leq t < n$, $r_3(t) \rightarrow r_{10}(0) \wedge r_{10}(1) \wedge \dots \wedge r_{10}(t)$. The approach also provides a very accurate formal model of modern constraint-based configuration approaches that use CSP (Constraint Satisfaction Problem) techniques (Wielinga & Schreiber, 1997).

The design-as-satisfiability approach precludes the need for performing a strict backward or forward chaining search (e.g., Coyne et al., 1990), and managing constraint satisfaction by a predefined control mechanism (as incorporated by some rule-based and expert systems for design¹⁶). In the SAT-based approach, all design-specific information is subsumed by simple uniform relationships between propositions, and the inference engine has no explicit indication as to what stands for a goal or what stands for a rule. This means that the system is not constrained to perform a strict backward or forward chaining search. This allows constraints to propagate more freely, and thus more quickly reduces the search space.

In logic and computer science, the SAT problem is well understood and analyzed. Thus, by mapping the synthesis problem to a logic representation, one can exploit the many well-known theorems on SAT that have been developed by researchers in the field to address interesting issues related to rule-based synthesis systems. Moreover, many researchers in different areas of computer science are developing

¹⁵We write $f(n) = O(g(n))$ if there are positive integers c and n_0 such that, for all $n \geq n_0$, $f(n) \leq cg(n)$.

¹⁶Some knowledge-based design systems use decomposition of large complex problems into smaller problems, and putting (recomposing) the subproblems together taking into account the interactions. The issue of recomposition is, then, dealt with as a constraint satisfaction (Maher, 1990).

faster SAT engines every year, which are freely shared and fine-tuned (e.g., Trick & Johnson, 1993). As a result, design systems can be created that combine the best features of these SAT engines.

The SAT-based approach is *fully* automated [rule-based design systems are often computer-assisted (e.g., Brown & Chandrasekaran, 1989)]. This automation would allow designers to work on many design problems simultaneously. Rather than progressing along a single search direction, designers could conveniently handle many alternatives without undue efforts. However, it is important to note that the SAT-based approach is not suggested as a monolithic architecture for design synthesis. A more useful architectural approach is to support *specific* synthesis tasks by means of computational shells—such as the SAT-based method—which help solve a portion of the design synthesis problem in a *computationally efficient* way. Furthermore, a SAT-based search engine can flexibly be plugged in various transformational-based synthesis systems.

The SAT encoding method has several weaknesses. The approach proposed in this article is most appropriate for problems where the optimum solutions (as in parametric design) will not be sought and satisfying solutions will fully be accepted. That is, instead of requiring an optimal design—which maximizes or minimizes an objective function of a number of requirements—we examine conceptual or preparametric design (e.g., Ulrich & Seering, 1989; Chakrabarti & Bligh, 1996) where designers accept a “good” or “satisfactory” solution. This is a strictly binary case for which propositional logic can be exploited. However, since propositional logic does not allow for variables, the proposed approach is (currently) limited when parameter values need to be set. Moreover, if the parameter values are taken from an unbounded domain or infinite domain, propositions cannot be fashioned.

In the future, we plan to perform extensive computational experiments for testing the performance of the SAT-based method on knowledge intensive designs—with respect to several algorithms for solving propositional satisfiability problems and various SAT encoding schemes (including analyzing the size of specific encoding schemes). We also plan to compare the new method with the currently available search engines (such as the inference mechanism presented in Sect. 2.2). Finally, since we are interested in reducing the number of variables and clauses as much as possible, we may consider several ways by which the Boolean expression can be simplified (e.g., by safely eliminating some of the axioms relating rules to their preconditions and effects).

REFERENCES

- Bradley, D.A., Bracewell, R.H., & Chaplin, R.V. (1993). Engineering design and mechatronics: The schemebuilder project. *Research in Engineering Design* 4(4), 241–248.
- Braha, D. (2000). *A new approach for synthesis*. Technical Report, Department of Industrial Engineering, Ben-Gurion University.
- Braha, D., & Brafman, R. (1998). *SAT-encoding for design search*. Technical Report, Ben-Gurion University.
- Braha, D., & Maimon, O. (1998). *A mathematical theory of design: Foundations, algorithms, and applications*. Boston: Kluwer Academic Publishers.
- Brown, K. (1997). Grammatical design. *IEEE Intelligent Systems* 12(2), 27–33.
- Brown, D. (1998). *Intelligent computer-aided design*. Technical Report, Computer Science Department, Worcester Polytechnic Institute.
- Brown, D., & Chandrasekaran, B. (1985). Expert systems for a class of mechanical design activity: Knowledge engineering. In *Computer-Aided Design*, (Gero, J., Ed.), pp. 259–283. Amsterdam: North-Holland.
- Brown, D., & Chandrasekaran, B., (1989). *Design problem solving: Knowledge structures and control strategies*. New York: Morgan Kaufmann Publishers.
- Brown, K., McMahon, C.A., & Sims Williams, J.H. (1994). A formal language for the design of manufacturable objects. In *Formal Design Methods for CAD, IFIP Transactions B-18*, (Gero, J.S. and Tyugu, E., Eds.), pp. 135–155. Amsterdam: North-Holland.
- Chakrabarti, A., & Bligh, T. (1996). An approach to functional synthesis of mechanical design concepts: Theory, applications, and emerging research issues. *Artificial Intelligence for Engineering Design, Analysis, and Manufacturing* 10(4), 313–332.
- Chandrasekaran, B. (1990). Design problem solving: A task analysis. *AI Magazine*, 11(4), 59–71.
- Chase, S.C. (1989). Shapes and shape grammars: From mathematical model to computer implementation. *Environment and Planning B*, 16(2), 215–242.
- Cook, S., & Mitchell, D. (1997). Finding hard instances of the satisfiability problem: A survey. *Proceedings of the DIMACS Workshop on Satisfiability Problems* 35, 1–17.
- Coyne, R.D., Rosenman, M.A., Radford, A.D., Balachandran, M., & Gero, J.S. (1990). *Knowledge-based design systems*. Reading: Addison-Wesley.
- Dym, C.L. (1994). *Engineering design—A synthesis of views*. Cambridge: Cambridge University Press.
- Ernst, M.D., Millstein, T.D., & Weld, D.S. (1997). Automatic SAT-Compilation of planning problems. *Proc. Fifteenth International Joint Conference on Artificial Intelligence*, Nagoya, Japan, 23–29.
- Fenves, S., & Baker, N. (1987). Spatial and functional representation language for structural design. In *Expert Systems in Computer-Aided Design*, pp. 511–529. Amsterdam: Elsevier.
- Fikes R., & Nilsson, N. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2(3–4), 189–208.
- Finger, S., & Rinderle, J.R. (1990). Transforming behavioral and physical representations of mechanical designs. *Proc. First Int. Workshop on Formal Methods in Engineering Design, Manufacturing, and Assembly*, 133–151.
- Frayman, F., & Mittal, S. (1987). COSSACK: A constraint-based expert system for configuration tasks. *Proc. Second Int. Conf. on Applications of AI to Engineering*, Boston, MA, 1–22.
- Fu, Z., De Pennington, A., & Saia, A. (1993). A graph grammar approach to feature representation and transformation. *International Journal of Computer Integrated Manufacturing* 6(1–2), 137–151.
- Garey, M.R., & Johnson, D.S. (1979). *Computers and intractability: A guide to the theory of NP-Completeness*. San Francisco: W.H. Freeman and Company.
- Gips, J., & Stiny, G. (1972). Shape grammars and the generative specification of painting and sculpture. In *Information Processing 71* (Freiman, C. V., Ed.), pp. 1460–1465. Amsterdam: North-Holland.
- Gips, J., & Stiny, G. (1980). Production systems and grammars: A uniform characterization. *Environment and Planning B* 7, 399–408.
- Gu, J. (1992). Efficient local search for very large-scale satisfiability problems. *Sigart Bulletin* 3(1), 8–12.
- Hoover, S.P., & Rinderle, J.R. (1989). A synthesis strategy for mechanical devices. *Research in Engineering Design* 1(2), 87–104.
- Huang, G.O., & Brandon, J.A. (1993). *Cooperating expert systems in mechanical design*. New York: John Wiley & Sons.
- Kannapan, S., & Marshek, K. (1990). An algebraic and predicate logic approach to representation and reasoning in machine design. *Mechanism and Machine Theory* 25(3), 335–353.

- Kautz, H., McAllester, D., & Selman, B. (1996). Encoding plans in propositional logic. *Proc. Fifth Int. Conf. on Knowledge Representation and Reasoning (KR-96)*, Boston, MA, 1–11.
- Kautz, H., & Selman, B. (1996). Pushing the envelope: Planning, propositional logic, and stochastic search. *Proc. Thirteenth Nat. Conf. on AI (AAAI '96)*, 1194–1201.
- Koo D.Y., Han S.-H., & Lee S.-H. (1998). An object-oriented configuration design method for paper feeding mechanisms. *Expert Systems with Applications 14(3)*, 283–289.
- Krishnamurti, R. (1992). The arithmetic of maximal planes. *Environment and Planning B: Planning and Design 19*, 431–464.
- Lee, C.-L., Iyenger, G., & Kota, S. (1992). Automated configuration design of hydraulic systems. In *Artificial Intelligence in Design '92* (Gero, J.S., Ed.), pp. 61–82. Dordrecht: Kluwer.
- Levesque, H., Mitchell, D., & Selman, B. (1992). A new method for solving hard satisfiability problems. *Proc. AAAI National Conference on Artificial Intelligence*, 440–446.
- Maher, M.L. (1988). HI-RISE: An expert system for preliminary structural design. In *Expert Systems for Engineering Design* (Rychener, M., Ed.), pp. 37–52. San Diego, CA: Academic Press.
- Maher, L.M. (1990). Process models for design synthesis. *AI Magazine 11(4)*, 49–58.
- Maimon O., & Braha, D. (1996). On the complexity of the design synthesis problem. *IEEE Transactions on Systems, Man, and Cybernetics 26(1)*, 142–150.
- Marcus S., Stout J., & McDermott, J. (1988). VT: An expert elevator designer that uses knowledge-based backtracking. *AI Magazine 9(1)*, 95–111.
- McDermott J. (1982). R1—A rule-based configurator of computer systems. *Artificial Intelligence 19(1)*, 39–88.
- McDermott J. (1993). R1 (XCON) at age 12: Lessons from elementary school achiever. *Artificial Intelligence 59(1–2)*, 241–247.
- Mitchell, T.M., Steinberg, L.I., & Shulman, J.S. (1985). A knowledge-based approach to design. *IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI 7(5)*, 502–510.
- Mittal, S., & Dym, C.L. (1986). PRIDE: An expert system for the design of paper handling systems. *IEEE Computer 19(7)*, 102–114.
- Mullins, S., & Rinderle, J.R. (1990). Grammatical approaches to design. *Proc. First Int. Workshop on Formal Methods in Engineering Design, Manufacturing, and Assembly*, 42–69.
- Pahl, G., & Beitz, W. (1988). *Engineering design: A systematic approach*. New York: Springer-Verlag.
- Sabin, D., & Weigel, R. (1998). Product configuration frameworks—A survey. *IEEE Intelligent Systems 13(4)*, 42–49.
- Schmidt, L., & Cagan, J. (1995). Recursive annealing: A computational model for machine design. *Research in Engineering Design 7(2)*, 102–125.
- Schreiber, A.Th., & Birmingham, W.P. (1996). The Sisyphus-VT initiative. Special issue of *International Journal of Human-Computer Studies*, 44(3/4).
- Selman, B., Kautz, H., & Cohen, B. (1996). Local search strategies for satisfiability testing. In *DIMACS Series in Discrete Mathematics and Theoretical Computer Science* (Johnson, D., & Trick, M., Eds.), vol. 26, pp. 521–532. American Mathematical Society.
- Selman, B., Kautz, H., & McAllester, D. (1997). Computational challenges in propositional reasoning and search. *Proc. Fifteenth Int. Joint Conf. on AI*, 50–54.
- Shin, H.-Y., & Lee, J.-W. (1998). Expert system for pneumatic design. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing 12(1)*, 3–11.
- Sriram, D. (1997). *Intelligent systems for engineering: A knowledge-based approach*. New York: Springer-Verlag.
- Suh, N.P. (1990). *The principles of design*. New York: Oxford University Press.
- Tommelein, I.D., Leit, R.E., Hayes-Roth, B., & Confrey, T. (1991). Sight-Plan experiments: Alternate strategies for site layout design. *ASCE Journal of Computing in Civil Engineering 5(1)*, 42–63.
- Tong, C., & Sriram, D. (Eds.). (1991). *Artificial Intelligence in Engineering Design*. New York: Academic Press.
- Trick, M., & Johnson, D. (Eds.). (1993). *Proc. DIMACS Challenge on Satisfiability Testing*. DIMACS Series on Discrete Mathematics. Piscataway, NJ: American Mathematical Society.
- Ulrich, K.T., & Seering, W.P. (1989). Synthesis of schematic descriptions in mechanical design. *Research in Engineering Design 1(1)*, 3–18.

- Wielinga, B.J., & Schreiber, A.Th. (1997). Configuration design problem solving. *IEEE Expert 12(2)*, 49–56.

Dan Braha was a Research Associate in the Department of Manufacturing Engineering at Boston University. Currently, he is an Assistant Professor in the Department of Industrial Engineering at Ben-Gurion University, Israel. He is also affiliated with the New England Complex Systems Institute in Cambridge, Massachusetts. He is spending 2001–2002 at the MIT Sloan school's Center for Innovation in Product Development (CIPD) as a Visiting Professor. One of his primary areas of research is engineering design and manufacturing. His research within engineering design focuses on developing methods to help the designer move from the conceptual phase to the realization of the physical device. He has developed a mathematical theory—the Formal Design Theory (FDT). Dan Braha has published extensively, including a book on the foundations of engineering design with Kluwer Academic Publishers, and a book on data mining in design and manufacturing, also with Kluwer. He was the editor of several special journal issues, and served as chair in several international conferences. His research interests include knowledge-based systems, case-based reasoning, cognitive science, artificial intelligence, computational complexity, information theory, product design and development, and operations research.

APPENDIX A. EXTENSIONS OF THE SAT-BASED APPROACH

In this appendix, we delineate several ways by which the approach can be extended.

A1. Comparing various encoding schemes

A number of encoding schemes are available for carrying out this approach, and one approach (i.e., linear encoding) is explicitly discussed. The same idea can be applied to all other encoding schemes.

A2. SAT encoding for predicate-based design knowledge

In this article, it is assumed without loss of generality that the SAT encoding technique is based on a design representation scheme (of the original design problem), which is expressible in propositional logic. The use of propositional logic is appropriate for modeling design knowledge such as: “**If** the car has OFF-HIGHWAY TIRE (r_{15}) **and** 4-wheel drive (s_1) **and** extra differential (s_{17}) **and** high ground clearance (s_{22}) **and** light weight (s_{27}), **then** the car is PASSABLE IN DIFFI-

CULT TERRAIN (r_{16}).¹⁷ However, in some design domains (such as machine design), it is not sufficient to model design knowledge by propositional logic alone. In such cases, we may use first-order logic, which is capable of expressing in much more detail than propositional logic a wide range of design knowledge.

In the following, it is briefly shown how to utilize the SAT encoding technique in case a first-order logic design representation scheme is employed.¹⁸ To illustrate, consider the domain of serial machine design, which is the process of transforming initial specifications of machine function into a serial arrangement of machine components (e.g., Schmidt & Cagan, 1995). First-order logic may be used to formalize machine design knowledge (rules), such as:

1. “**If** assembly_2 converts reciprocating rotational energy into continuous translational energy **and** assembly_3 converts continuous translational energy into continuous electrical energy **and** assembly_2 is connected to assembly_3, **then** assembly_1¹⁹ converts reciprocating rotational energy into continuous electrical energy”;
2. “**If** a rack & pinion *represents* assembly_1, **then** assembly_1 converts reciprocating rotational energy into reciprocating translational energy.”

In order to formalize such design knowledge in a first-order logic, we consider (1) a *finite* set of *constants*—*gear*, *coil_spring*, *transformer*, *electric motor*, . . . , s_1, s_2, \dots —for representing machine components (e.g., transformers, torsional springs, and gears) and machine assemblies;²⁰ and (2) a finite set of *predicate names*— Q_1, Q_2, \dots —for expressing machine functions (e.g., convert reciprocating rotational energy into reciprocating translational energy), R_1 for expressing an assembling of assembly_i connected to assembly_j, and R_2 for expressing the representation of an assembly by a (concrete) machine component. For example, the design rules stated above are formalized as the following rule schemas:

$$(Q_2(s_2) \wedge Q_3(s_3) \wedge R_1(s_2, s_3)) \rightarrow Q_1(s_1)$$

$$R_2(\text{rack \& pinion}, s_1) \rightarrow Q_4(s_1)$$

Since all of the constants that appear in the definition of the first rule schema are s_1, s_2, s_3 , we denote the rule as $p_1(s_1, s_2, s_3)$. Similarly, we denote the second rule schema as $p_2(\text{rack \& pinion}, s_1)$.

¹⁷As in Section 2.1, this rule is represented in propositional logic as $s_1 \wedge s_{17} \wedge s_{22} \wedge s_{27} \wedge r_{15} \rightarrow r_{16}$.

¹⁸For a more elaborate presentation of the first-order logic design representation scheme see Braha (2000).

¹⁹Assembly_1 represents the arrangement of assembly_2 connected to assembly_3.

²⁰By arranging the assemblies in hierarchy, we guarantee that each assembly is used in the design process no more than once; see Braha (2000).

Having introduced the basis of the first-order logic representation scheme, we briefly describe how the encoded SAT problem is obtained from this representation.²¹ We have to specify two things: the set of propositions used, and the set of expressions (or axioms) to be generated. Let n be the bound on the number of design process steps. In the machine design domain, we will use propositions such as $R_2(\text{rack \& pinion}, s_1, t)$ to mean that a rack & pinion instantiates assembly_1 at time t ; $R_1(s_1, s_2, t)$ to mean that assembly_1 is connected to assembly_2 at time t ; $Q_1(s_1, t)$ to mean that assembly_1 converts reciprocating rotational energy into continuous electrical energy at time t ; and $p_2(\text{rack \& pinion}, s_1, t)$ to mean that a rack & pinion instantiates assembly_1 between times t and $t + 1$. As before, in the *design-as-satisfiability* approach, design is simply a set of axioms with the property that *any* model (i.e., a satisfying truth assignment) of the axioms corresponds to a valid design solution. For the simple machine design specification of transforming the function “converting reciprocating rotational energy into continuous electrical energy” (denoted as the predicate Q_1) into an arrangement of machine components, we shall need the following initial and goal state axioms, abbreviated as schemas. A schema stands for the set of all the formulas that can be generated by iterating the quantifiers over the constants of the appropriate types. A universally quantified expression expands to the *conjunction* of its instantiations. For instance, $\neg Q_1(s_2, 0) \wedge \neg Q_1(s_3, 0) \wedge \neg Q_1(s_4, 0) \wedge \dots \wedge \neg Q_1(s_k, 0)$ is an instance of the schema $\forall x \neq s_1 (\neg Q_1(x, 0))$.

INITIAL STATE AXIOM:²²

$$Q_1(s_1, 0) \wedge \forall x \neq s_1 (\neg Q_1(x, 0)) \wedge \forall x (\neg Q_2(x, 0)) \wedge \dots \\ \wedge \forall x (\neg S_1(x, 0)) \wedge \forall x \forall y (\neg S_2(x, y, 0)) \quad \blacksquare$$

GOAL STATE AXIOM:²³

$$\forall x (\neg Q_1(x, n)) \wedge \forall x (\neg Q_2(x, n)) \wedge \dots \wedge \forall x (\neg Q_k(x, n)) \quad \blacksquare$$

The other axioms, which describe the production rules in general, are specified in a similar manner as discussed in Example 1. For example, the following axiom rules out the possibility that the production rule $p_1(s_1, s_2, s_3)$ is executed despite the fact that its preconditions are false:

$$p_1(s_1, s_2, s_3, t) \rightarrow (Q_1(s_1, t) \wedge \neg Q_2(s_2, t + 1) \wedge Q_3(s_3, t + 1) \\ \wedge S_1(s_2, s_3, t + 1)).$$

In summary, we showed how to utilize the SAT encoding approach in solving design problems that are expressible in first-order logic. The first-order logic contains a *finite* col-

²¹Although we concentrate on the linear encoding technique, we note that other more compact SAT encodings are possible for the first-order design representation scheme.

²²Recall that the initial state is completely specified. The goal state may be either fully or partially specified.

²³That is, there will be no unmatched machine functions, Qs , in the final process state.

lection of nonlogical symbols, predicates without variables (such predicates are called *ground atoms*), and production rules without quantifiers (i.e., without \forall, \exists). Indeed, we can view the above (“finite”) first-order logic as an applied propositional logic, by letting the *ground atoms* of the first-order language play the role of propositional variables. From this point of view, the encoded set of axioms (or SAT encoding) can be interpreted as an *expression in propositional logic, using a truth assignment for its ground atoms*.

A3. Embedding domain-specific search control knowledge

In Example 1, we showed how to encode structural constraints such as: “structural attributes s_1 and s_2 cannot be included together in a physically realizable design.” Other domain-specific constraints may be considered as well. For example, according to Suh (1990), in an acceptable design, the structural attributes and the functional attributes are related in such a way that a specified structural attribute can be adjusted to satisfy its corresponding functional attribute without affecting other “conflicting” functional attributes.²⁴ This principle can be implemented in the SAT-based encoding as follows. Assume that the functional attributes r_{21} (“safety in flipping over”) and r_{10} (“high driving speed”) are conflicting *with respect to* the structural attribute s_{27} (“light weight”). We shall need “local decoupling” axioms, which say that if structural attribute is part of a design solution, then either functional attribute r_{21} or r_{10} (but *not* both) is decomposed at any time point. That is, for every $t_1, t_2 \leq n$ we add:

$$s_{27}(n) \rightarrow \neg r_{21}(t_1) \wedge \neg r_{10}(t_2).$$

APPENDIX B. A SAMPLE OF RULES FOR THE AUTOMOBILE EXAMPLE

RULE 1.

IF: The car has OFF-HIGHWAY TIRE **and** 4-wheel drive **and** extra differential **and** high ground clearance **and** light weight
 THEN: The car is PASSABLE IN DIFFICULT TERRAIN ■

RULE 2.

IF: The car is SAFE IN POOR EXTERNAL CONDITIONS

THEN: The car is SAFE **and** the external conditions are **not** good ■

RULE 3.

IF: The car is SAFE IN HIGH DRIVING SPEED
 THEN: The car is SAFE **and** the maximum allowed speed is 200 Km/h ■

RULE 4.

IF: The car is SAFE IN ACCIDENTS
 THEN: The car is SAFE **and** the car is used for family driving **and** the external conditions are good **and** the maximum allowed speed is 160 Km/h ■

RULE 5.

IF: The car is SAFE IN POOR VISIBILITY **and** SAFE IN BAD WEATHER **and** SAFE IN OFF-HIGHWAY ROAD
 THEN: The car is SAFE IN POOR EXTERNAL CONDITIONS ■

RULE 6.

IF: The car is SAFE IN HEAD-ON COLLISIONS **and** SAFE IN SIDE COLLISIONS **and** SAFE IN FLIPPING OVER **and** has automatic belts
 THEN: The car is SAFE IN ACCIDENTS ■

RULE 11.

IF: The car has absorbent front end **and** air bag **and** an engine that deflects down
 THEN: The car is SAFE IN HEAD-ON COLLISIONS ■

RULE 12.

IF: The car has extra strong door
 THEN: The car is SAFE IN SIDE COLLISIONS ■

RULE 21.

IF: The car has AERODYNAMIC DESIGN **and** EFFICIENT ENGINE **and** DIESEL ENGINE **and** disconnecting fan system **and** light weight
 THEN: The car has LOW FUEL CONSUMPTION ■

RULE 22.

IF: The car has tubeless tire **and** radial tire
 THEN: The car has RELIABLE TIRE ■

RULE 38.

IF: The car has AERODYNAMIC DESIGN **and** EFFICIENT ENGINE **and** DIESEL ENGINE **and** disconnecting fan system **and** tire with 155 width symbol **and** tire with 60 aspect ratio symbol
 THEN: The car has LOW FUEL CONSUMPTION ■

²⁴This principle is termed in Suh (1990) as the “independence axiom.”