MASSACHUSETTS INSTITUTE OF TECHNOLOGY
ARTIFICIAL INTELLIGENCE LABORATORY

Working Paper 309 February 1987

A Partial Mechanical Design Compiler

Allen C. Ward

Abstract

I have implemented a simple "mechanical design compiler", that is a program which can convert high-level descriptions of a mechanical design into detailed descriptions. (Human interaction is sometimes required.) The program operates in the domain of power transmission equipment composed of discrete, purchasable components. I describe a semantic theory which assigns meanings to the high-level descriptions, and a set of operations on statements in a "specification language" which perform some of the reasoning required by the "compilation" process.

1 Introduction

Most of the computational tools available to mechanical designers directly support drawing or analysis; they support design only indirectly. Automating some design decisions would improve designer productivity, prevent errors, and allow the exploration of more alternatives in greater depth.

There are roughly 30,000 categories of mechanical equipment listed in the Thomas Register; it would be expensive to write a special purpose program to help us design each kind of equipment. Rather, we need high level design languages, which allow the designer to describe the design in convenient terms. **Compilers**[1] for these languages should "fill in the details", forming descriptions detailed enough that appropriate components can be made or purchased.

These high level languages should be **compositional**. That is, the elements of the language should themselves be "designs" with the same status as the descriptions produced by combining the elements, and it should be possible to further combine newly formed designs. Figure 1 shows one of the many possible combinations of motor, load, and transmission symbols in a "schematic" language.

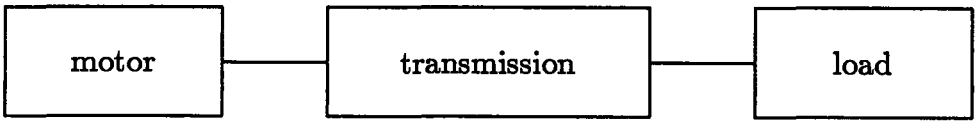


Figure 1: A power train schematic

Special purpose programs can be tuned to work properly under narrowly defined conditions, but compositional languages must work correctly even when assembled into systems not foreseen by the programmer. This seems achievable only if the program performs a reasonable number of clearly defined types of operations, such that we can be confident that each type will be executed only when that execution is correct. This means that we need to assign clear meanings to the symbols of our design description language;

[1] I will introduce terminology in **bold face**, and specific claims of my model in *italic*.

express the compiler's manipulations of these symbols in (usually mathematical) forms compact enough to follow and reason about; and relate the manipulations clearly to the real world. That is, we need a theory.

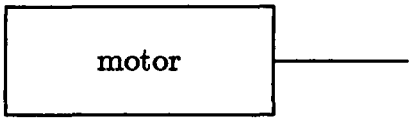
My research develops such a computationally adequate theory for part of mechanical design. Part of the theory is quite general, part specific to the design of constant and variable speed power trains using components from catalogs. In the next section I discuss how we can assign meaning to the descriptions used by designers. In section 3 I introduce some extensions to the traditional mathematical language of mechanical engineering. These extensions enable me to compactly define the operations of a simple design compiler. The operation set is not yet complete; that is, the program often requires human interaction to complete the design.

2 The Meaning Of Partially Completed Designs

Consider schematics, part drawings, specifications, catalog descriptions, or the structure of 1s and 0s in a computer's memory which represent a solid model or the program for machining a part. Each is an arrangement of symbols. Knowledgeable experts or suitable machines may take each as input, and perform sequences of **operations** which ultimately produce physical objects, or **artifacts**. We might think of these experts and machines as completing a design, and I will call the symbol arrangements **partially completed designs**.

Considered abstractly, design programs can work only by changing partially completed designs, in ways determined entirely by the pattern of the symbols. But how can we determine if these changes are correct? The partially completed designs are only symbol patterns; to reason about them, we need to understand precisely what things in the real world the symbol patterns represent.

Partially completed designs will be seen differently from different perspectives. Consider the follow schematic symbol.



To a purchasing agent, the symbol represents every purchasable motor. We could say that the operation of "buying", applied to the symbol, defines a set of motors, which includes every purchasable motor.

To a designer, however, the symbol represents a somewhat different set. He does not yet know the part number needed to order the motor he wants. However, if he knows the motor will provide less than 3 horsepower, he assumes it will weigh less than 90 pounds. If he has not yet picked a transmission, he does not know how much torque the motor will need to supply, but he assumes in making other decisions about the design that the motor will supply adequate torque. We could say that he is reasoning about the design operations he will perform, or alternatively, that he is reasoning about the set of particular motors (artifacts) defined by those operations.

Note that he also reasons about the **environments** (the possible transmissions and loads) in which the motors might be used. In my model, *partially completed designs represent sets of sequences of design and manufacturing operations, or equivalently the sets of artifacts which those operations might produce. They also represent the set of environments in which these artifacts might be used. We can evaluate the changes design programs or human designers make in partially completed designs by thinking about these sets.*

3 Representing Power Trains

Note that because of manufacturing tolerances, even so detailed a description as "Dayton motor 2M167" designates any one of an infinite number of slightly different motors. We cannot in general assume a *priori* that these differences do not matter; for example, there is no guarantee that the half-inch shaft of a particular motor will fit properly into a particular coupling. We much consider the range of possibilities represented by this set of motors.

The schematic motor symbol represents the union of many such sets.

We can now see one reason that analysis has been easier to automate than design. Physics provides mathematical tools for representing and reasoning about *particular* physical objects. Mechanical design requires reasoning about infinite sets of different physical objects.

We can easily form qualitative statements about infinite sets, e.g. "for equivalent horsepower, hydraulic motors are lighter than electric motors". Such statements are however too vague for reliable, compositional design languages. We need new mathematical tools for **quantitative** reasoning about sets of artifacts. I have developed such tools for simple power trains; the following sections give the flavor, rather than a summary, of that work.

I use three kinds of representation for power trains. First, part numbers (with attached data) from catalogs represent the obvious sets of purchasable artifacts. I will loosely call both the part number entry and the set of artifacts it designates a **component**. Lists of part numbers represent the union of such sets.

Second, such lists are assigned to schematic icons, which display connections. The connected motor and transmission pair in Fig. 1 represent the set formed by connecting, in the obvious way, each represented motor with each represented transmission. The load in this case represents the environments of use.

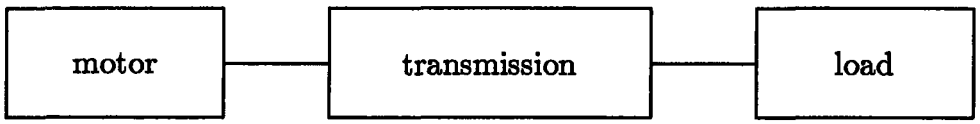
Third, the **specification language** discussed below is used to describe user specifications, components, the super-sets represented by lists of part numbers, and the sets represented by linked icons.

These three representation forms can be viewed as projections of or perspectives on the artifact sets. The designer can specify a design by connecting icons to form power trains, and providing specifications. Formally defined compiler operations derive specification language descriptions characterizing the set of artifacts represented, and apply specifications imposed by the user and by connected component sets to eliminate unsuitable part numbers from the catalog lists.

3.1 The Specification Language

Consider the torque, RPM, horsepower, and transmission ratio characteristics we need to represent in order to convert the schematic below into an

actual motor and transmission part number.



The motor may need to supply 3 horsepower. This specification is a statement about the environment in which the motor is to be used. But the transmission is part of that environment; for easy compositionality, specifications and artifact set descriptions should be in the same language.

The specification language uses equations and intervals on the real number line, but adds some notions. Note that the load might **require** that any transmission connected to it be able to supply torques of up to 20 newton-meters. On the other hand, every motor in the set represented might **assure** the ability to supply up to 2 newton-meters. Each of these is a statement about a variable and a range of values, but they are quite different in meaning.

Also, while the load requires the supply of torques up to 20 newton-meters, it may also require that the torque never exceed 30 newton-meters. The first of these statements discusses a **range of operation**, the second **limits of variation.**

Using a pair of square brackets [] to represent limits of variation, and a double headed arrow $\leftrightarrow$ to indicate range of operation, we can write such statements compactly. For example, in describing a set of motors, ⟨[] RPM 1650 1750 Assured⟩ means that every motor in the set will keep the speed between 1650 and 1750 rpm. ⟨$\leftrightarrow$ Input-torque 0 2 Required⟩ might express that each of a set of transmissions needs input torques ranging from 0 to 2 newton-meters.

4 Some Useful Operations

4.1 Abstracting Aggregated Sets

Given a list of components, with associated descriptions in the specification language, we can form descriptions of the set of artifacts represented

by the whole list. The range of operations, assured or required, is the intersection of the ranges of operation of the components. The possible variation is formed by taking the highest upper and lowest lower bounds of the component possible variations; this is looser than taking the union.

These operations make intuitive sense in terms of the meaning ascribed to the descriptions; the resulting statements also produce appropriate results when the following operations are applied.

4.2 Eliminating Unsatisfactory Implementations

Initial partially completed designs represent a wide variety of artifacts. Design operations are needed to narrow the sets represented.

For example, if ⟨↔ RPM 1500 1800 Required⟩ for every implementation, a component with ⟨[] RPM 1600 1700 Required⟩ can be eliminated from consideration. Generally, if for a set of artifacts we have ⟨↔ x A B ...⟩ and ⟨[] x C D ...⟩ and the interval $[A\ B]$ is not a subset of $[C\ D]$, we can conclude that none of the artifacts will be satisfactory.

I have defined three other elimination operations. The correctness of each depends on the fact that as parts are eliminated ranges of operation can only get bigger, possible variations only smaller.

4.3 Using Equations To Characterize Composed Sets

Designers (and design programs) usually cannot reason about an entire design at once; designs are too complex. *Designers often need to isolate a sub-design, form specifications for the sub-design from information about its potential environments (the rest of the design), then reason locally using those specifications.*

Isolating sub-designs requires abstracting information about compositions of partially completed designs. For example, descriptions of the load in Fig. 1, e.g. ⟨↔ torque 0 20 Required⟩ cannot be applied directly to the motor. We must abstract the load-transmission pair. We use the transmission equation Output-torque = Input-torque × Ratio. If available ratios include 2:1, 3:1, and 5:1, we can conclude ⟨↔ input-shaft-torque 0 4 Required⟩ for the set of transmissions.

Generally, if for a set of artifacts we have $\langle \leftrightarrow\ x\ A\ B \ldots \rangle$, and $\langle [] \ y\ C\ D \ldots \rangle$, and if $y = f(x, z)$ where z and x vary independently, we can conclude

$\langle [] \ z\ U\ V$ Required$\rangle$
where
$U = \min(z)$ such that for all x in $[A\ B]$ y is in $[C\ D]$ and
$V = \max(z)$ such that for all x in $[A\ B]$ y is in $[C\ D]$.

This inference is among the more complex of five used for abstracting composed designs.

5 Further Work

First, the representation and operations need to be more thoroughly tested. I have not yet used the system to perform a large-scale design.

Second, when the artifact sets include a wide range of devices, my specification language descriptions of them are loose. *Descriptions of diverse artifact sets are often too weak to support reasoning completing the design. Designers must then explore multiple sub-sets.* For efficiency, this **search** needs to by guided by a utility function, so that it finds the best rather than every feasible design. Adding search to the compiler will make it more autonomous.

Third, the specification language operations thus far implemented are not complete.

Fourth, the limits of the system need to be defined. For example, it is clear that there a kinds of design that the specification language cannot represent; even the closely related field of servo-system design is beyond its current capabilities.

6 Related Work

Simon[1] pointed out the "compositional" nature of design, while Clancey[2] observes that the "expert systems" approach is poorly suited to such compositional problems. Dixon et al [3] compose fully dimensioned geometric "features"; they evaluate but do not refine the composed descriptions.

Ulrich (private communication) automatically composes "functional elements", then uses qualitative reasoning to "debug" the resulting designs by combining functional elements or adding new ones.

Silicon compilers operate on composed descriptions in the electronic domain. Mitchel et al[4] have written a program which successively modifies compositional partially completed electronic designs, and checks qualitative constraints between parts of the designs to select between alternatives.

Taguchi [5] explicitly (but heuristically and informally) uses simulations of or experiments with particular artifacts to reason about sets of artifacts; he tries to select the most reliable of a set of alternative partially completed designs.

Mittal et. al.[6] and Brown[7] use special purpose (non-compositional) trees of design operation sequences to refine partially completed designs, for paper feed paths and air cylinders, respectively.

Chapman's[8] "partially completed plans" suggested ideas and terminology; if interpreted in Chapman's style, each partially completed design would represent all those partially completed designs which might be derived from it by applying design operations.

Serrano and Gossard [9] provide mechanisms for propagating algebraic constraints; these can perform some of the operations defined on my specification language.

References

[1] H. A. Simon. *The Sciences of the Artifical.* The MIT Press, 1981.

[2] William J. Clancey. Heuristic classification. *Artificial Intelligence*, 27, 1985.

[3] J. R. Dixon, E. C. Libardi Jr., S. C. Luby, M. Vaghul, and M. K. Simmons. Expert systems for mechanical design: examples of symbolic representations of design geometries. In *Applications of Knowledge-based Systems to Engineering Analysis and Design*, ASME, New York, NY, 1985.

[4] T. M. Mitchell, L.I. Steinberg, and J.S. Shulman. *VEXED: A*

Knowledge-based VLSI Design Consultant. Technical Report LCSR-TR-57, Rutgers University, 1984.

[5] Genichi Taguchi. *Introduction to Quality Engineering.* UNIPUB, White Plains, NY, 1986.

[6] S. Mittal, C. L. Dym, and M. Morjaria. Pride: an expert system for the design of paper paths. In *Applications of Knowledge-based Systems to Engineering Analysis and Design*, ASME, New York, NY, 1985.

[7] David Brown. Capturing mechanical design knowledge. In *Proceedings of the 1985 ASME International Computers in Engineering Conference*, ASME, Boston, MA, 1985.

[8] David Chapman. *Planning for Conjuntive Goals.* Technical Report 802, MIT Artificial Intelligence Lab, 1985.

[9] David Serrano and David Gossard. Constraint management in conceptual design. In *Knowledge Based Expert Systems in Engineering: Planning and Design*, Computational Mechanics Publications, 1987.