Conceptual Design and Artificial Intelligence*

Devika Subramanian
Department of Computer Science
Cornell University
Ithaca, NY 14853
devika@cs.Cornell.edu

Abstract

This paper introduces new approaches to the conceptual design of electro-mechanical systems from qualitative specifications of behaviour and function. The power of these methods stems from the integration of techniques in qualitative physics, symbolic mathematics, computational geometry and constraint programming. This is illustrated with an effective kinematic synthesis method that integrates reasoning with configuration spaces and constraint-programming techniques.

1 Introduction

The broad goal of our research is to derive computational theories of conceptual or pre-parametric design. As manufacturing technologies change, as new materials are developed, as new design constraints emerge (designs with recyclable parts, and designs that assemble and disassemble easily), as products become more complex, as the need to build in continuous improvement into design processes emerges, basic conceptual design procedures for electro mechanical systems require broadening with effective use of computer tools in the early stages of design. Our specific aim is to use methods from artificial intelligence, especially qualitative physics and constraint programming, with techniques from computational geometry and symbolic mathematics to build new computational prototyping tools for conceptual design.

2 A Case Study

While the talk will present several case studies of effective tools for conceptual design in a variety *of* domains, the rest of this paper is devoted to an illustration in the context of mechanism synthesis[1]. Mechanisms are an important part of most electro-mechanical systems. They transmit motion from one rigid body to another. Our design system takes as input constraints on the *motion* of a mechanism in qualitative, mathematical form. As output, it produces a *systematic enumeration* of mechanism topologies and geometries that satisfy the given constraints. Tt also performs high-level simulation to demonstrate the feasibility of the design. The conceptual designs produced by our system can be refined and optimized by constraint-solving systems that select candidate designs based on cost, material, manufacturing and assembly constraints.

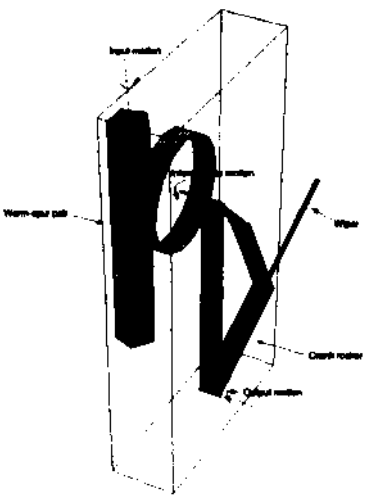

Figure 1: Design of a windshield wiper

The running example used in this paper is the synthesis of a windshield wiper whose input power is provided by a motor rapidly rotating around the *z* axis and whose output is an oscillation in the *yz* plane with low frequency. Note that this a *partial* description of the input and output *motions* of this device.

The first design enumerated by our system is shown in Figure 1. It employs a worm spur which converts the uniform input rotation around the *z* axis to one about the *x* axis. The output of the spur gear drives a crank rocker. The overall output is tapped from the rocker. Dimensions, positions and orientations of the gears and

*This work is supported by NSF-IRI-8902721, the Xerox Design Research Institute, and the Moore Fund.

*This is joint work with my student Cheuk-san (Edward) Wang.

the crank rocker are calculated by the system. Another design generated by our system satisfies the same motion specifications using a rack and gear pair, where the rack is driven by a slider crank with the crank being rotated uniformly by a worm spur pair. The worm itself is connected, in both cases, to a motor shaft.

There are several unique aspects of our method. We have a uniform representation for constraints, and can take them into account during the synthesis process. In this example, motion constraints as well as dimensional constraints are handled simultaneously. The synthesis process is very efficient. The example synthesis above was generated in about half a second on a Sparc station. Relevant constraints are enforced as soon as they become applicable. This is what makes the generation process efficient: we elaborate this point in Section 7.

The synthesis process is grounded in a mathematical theory of motion composition that is based on configuration spaces. We compile the algebraic theory of motion synthesis into a qualitative form that preserves essential distinctions for the specification and solution of a large class of kinematic synthesis problems. We introduce a property called *join preservation* which is a constraint on a qualitative motion language that is needed to guarantee the generation of correct designs. Our synthesis algorithms are actually implemented and are currently being field tested at the Xerox Webster Design Research Center. Our program has produced innovative designs for a number of common devices described in [19]. An interesting set of egg-beater designs in shown in Figure 4.

2.1 The Problem: Motion Synthesis

We now describe the synthesis problem addressed in this paper in detail. Kinematic synthesis is the problem of determining a three dimensional structure of rigid parts that implements a given motion specification. Kinematics only considers motions and not the forces that cause the motions. Conceptual kinematic synthesis is generally acknowledged to be a very difficult problem. A modern textbook in the area [18] states that

> The designer generally relies on intuition and experience as a guide to type and number synthesis. Very little supporting theory is available in these areas.

Conceptual synthesis of mechanisms is difficult because designs are typically specified in incomplete terms and by their intended use (e.g., a fruit-picker or a fuel-hose connector). There is no general theory that relates function and structure in mechanical devices. That is, the space of mechanisms that achieves a given functional specification is not exhaustively and systematically enumerable. Compendia such as Artobolevsky's catalog [1] provide a library of known mechanisms indexed by type (lever mechanisms, e.g.) and function (e.g., indexing). They are a useful starting point for a designer who can then use systematic adaptation of these designs to create devices which meet the specified functionality. The derivation of the motions that accomplish a given function is an open problem that is not addressed in this paper. Given the motions, we call the problem of designing a structure that generates them, the *motion synthesis* problem. This is also difficult to solve as it involves deriving geometry from motion. Most of the current work on conceptual design of mechanisms focuses on this problem [6, 11, 14, 15, 16, 26].

Previous work in this area falls under three major categories: structural, behavioural and functional theories of synthesis. *Structural* theories[6] generate mechanism topologies systematically, usually from specification of the number of links and the total number of degrees-of-freedom of the mechanism. Pure structural theories of synthesis result in a generate-and-test method for producing mechanisms given input-output motion specifications. This procedure is usually quite expensive, and it is generally difficult to exploit information about the desired motion to control the enumeration phase. *Behavioural* theories derive the structure of a mechanism from specifications of its output and input motions. They fall into two categories: compositional and non-compositional. *Compositional* theories of motion synthesis [16, 24, 9, 5], assume the presence of primitive or atomic building blocks which implement simple input-output motion specifications. They provide methods for systematically breaking down a complex input-output specification in terms of the primitive ones. Compositional theories typically address the synthesis of large scale electro-mechanical systems. *Non-compositional* theories [11, 14] build structures "from scratch" that satisfy given motion specifications. They are typically used for small-scale electro-mechanical systems or the synthesis of specific parts. *Functional* theories are theories that work from intended function of a device and derive a structure that performs that function. Functional theories of synthesis posit intermediate behavioural specifications and then design structures that generate those behaviours. Few functional theories exist in the literature: [7] and [10] take kinematic function into account in the design of mechanism topologies.

This paper is organized as follows. In Section 3, we formally define motion specifications in terms of configuration spaces and introduce abstract and concrete mechanisms. The operators which compose abstract mechanisms and their concrete counterparts are presented in Section 4. The composition operators form the basis for a rigorous specification and solution of the motion synthesis problem in Section 5. In Section 6, we discuss tractable qualitative representations of the configuration space descriptions manipulated by our algebraic synthesis method. We then present efficient constraint solving algorithms that use the qualitative representations. These algorithms have been implemented in CLP(R) [27] and we present examples of interesting syntheses in Section 7. We conclude by reiterating the main contributions of our paper and provide a discussion of future work on the problem of automating motion synthesis, and more generally on the problem of conceptual design.

3 Configuration Spaces, Motions, and Mechanisms

We briefly review the concept of a configuration space before formally defining the motion of an object. Let

F_A be a Cartesian frame embedded in the object A, and let F_W be the fixed frame. The origin O_A of F_A is the reference point on A.

Definition 1 *A configuration of A is a specification of the position and orientation of F_A with respect to F_W. The configuration space of A is the space C of all possible configurations of A.*

The motion M of an object is a description of how its configuration changes with time. The configuration of a planar rigid body undergoing pure rotation at time t, is $(x_A, y_A, \theta(t))$ where x_A and y_A are fixed. (x_A, y_A), the center of rotation, is also the origin of F_A, and $\theta(t)$ is the angular orientation of F_A with respect to F_W, at time t.

Definition 2 *The motion of an object is a continuous function from time to its configuration space.*

Now we are ready to formally define a mechanism. Reuleaux [23] defines a mechanism as "a combination of rigid bodies so formed and connected that they move upon each other with definite relative motion." We distinguish between an *abstract mechanism* which is a relation on the configuration spaces of the input and output links, from a *concrete mechanism* which is a 3D arrangement of rigid bodies that implements this relation. The difference between an abstract and concrete mechanism is that while an abstract mechanism describes *what* a mechanism does, the concrete mechanism is a specification of *how* it does it.

Definition 3 *An abstract mechanism is a relation $\mathcal{R}$ which is a subset of $I \times O$ where I and O are configuration spaces of its input and output links respectively.*

Consider a meshed gear pair A and B on the xy plane with fixed centers at (x_A, y_A) and (x_B, y_B). $\alpha > 0$ is the gear ratio, and ϕ is the initial difference in angular position between the local reference frames attached to the centers of the two gears. It implements the relation $\mathcal{G} \subseteq I \times O$, $I = \{(x_A, y_A, \theta) \mid \theta \in [0, 2\pi)\}$ and $O = \{(x_B, y_B, \psi) \mid \psi \in [0, 2\pi)\}$,

$$\begin{aligned} \mathcal{G} \;=\; & \{(x_A, y_A, \theta, x_B, y_B, \psi) \mid \psi = -\alpha\theta + \phi, \\ & (x_A, y_A, \theta) \in I, (x_B, y_B, \psi) \in O\} \end{aligned} \quad (1)$$

This description[2] has abstracted structure and could well be implemented by any mechanism that transforms uniform unconstrained rotation around the z axis at (x_A, y_A) to uniform unconstrained rotation of the opposite sense with angular velocity scaled by α around the z axis at (x_B, y_B). Note that $\mathcal{G}$ has only one degree of freedom, namely θ. All other components of this relation are either constants or can be calculated from θ.

Definition 4 *The degree of freedom (DOF) of the abstract mechanism $\mathcal{R}$ is the dimension of $\mathcal{R}$.*

The definition of an abstract mechanism provided above, is precise, but sometimes unintuitive. It is hard to understand the behaviour of a mechanism as a general algebraic relation on multi-dimensional configuration spaces. This motivates us to define an abstract mechanism as a set of pairs of input and output motions.

[2]We have omitted the relation between center distance and pitch diameter of the gears for simplicity.

Definition 5 *Let $\mathcal{R}$ be an abstract mechanism satisfying Definition 3, and let M_i and M_o be motions of the input and output links respectively. Then this abstract mechanism can be defined alternatively as $\mathcal{R}'$, where*

$$\mathcal{R}' = \{(M_i, M_o) \mid \forall t\, (M_i(t), M_o(t)) \in \mathcal{R}\}$$

While $\mathcal{R}$ is a relation on configuration spaces and relates the instantaneous positions of the input links with those of the output links, $\mathcal{R}'$ is a relation between two motions.

The implementation of an abstract mechanism—a concrete mechanism, is a geometric description of links and joints between links. This description is stored as a kinematic diagram annotated with a set of constraints.

Definition 6 *A concrete mechanism is a pair (K, C) where K is the kinematic diagram of the mechanism and C is a set of geometric and dimensional constraints on K. There are two distinguished classes of links in K : input links I, and output links O. The relation between the configuration spaces of the input and output links of (K, C) is its corresponding abstract mechanism.*

Following Freudenstein [8], we use graph theory to formally specify a kinematic diagram. The kinematic graph K of a concrete mechanism is an undirected graph (V,E), where V is the set of vertices that denote the links, and the edge set E represents the kinematic pairs. The edges are labeled according to joint type [23]. There are one or more nodes in V standing for the input and output links of the mechanism. An annotated kinematic graph has algebraic constraints on elements of V and E represented as a constraint set C.

4 Composition of Mechanisms

Complex mechanisms are composed out of simpler ones. During composition, we construct the relation (I_1, O_2) from (I_1, O_1) and (I_2, O_2) by imposing the equality constraint $O_1 = I_2$. Two configuration spaces are equal if and only if they denote the same set of configurations. To enforce the equality constraint between two configuration spaces, we intersect them to obtain the configurations common to both sets.

Definition 7 *The composition $\bowtie$ of two abstract mechanisms $\mathcal{R}_1 \subseteq I_1 \times O_1$ and $\mathcal{R}_2 \subseteq I_2 \times O_2$ is*

$$\mathcal{R}_1 \bowtie \mathcal{R}_2 = \{(i_1, o_2) \mid o_1 = i_2 \wedge (i_1, o_1) \in \mathcal{R}_1 \wedge (i_2, o_2) \in \mathcal{R}_2\}$$

An example of composition of two abstract mechanisms is illustrated with the construction of $\mathcal{GT}$ that can be implemented by a gear train, from two relations $\mathcal{G}_1$ and $\mathcal{G}_2$ that can be implemented by gear pairs. We use the relation in Equation 1 to define $\mathcal{G}_1$ and $\mathcal{G}_2$.

$$\begin{aligned} \mathcal{G}_1 &= \{(x_A, y_A, \theta, x_B, y_B, \psi) \mid \psi = -\alpha_1\theta + \phi\} \\ \mathcal{G}_2 &= \{(x_M, y_M, \theta', x_N, y_N, \psi') \mid \psi' = -\alpha_2\theta' + \phi'\} \\ \mathcal{GT} &= \mathcal{G}_1 \bowtie \mathcal{G}_2 \\ &= \{(x_A, y_A, \theta, x_N - x_B, y_N - y_B, \psi') \mid \\ &\quad (x_A, y_A, \theta, x_B, y_B, \psi) \in \mathcal{G}_1, \\ &\quad (x_M, y_M, \theta', x_N, y_N, \psi') \in \mathcal{G}_2, \\ &\quad (x_B, y_B, \psi) = (x_M, y_M, \theta')\} \end{aligned}$$

The gear ratios α_1 and α_2 are both positive. From $\mathcal{GT}$, we can derive the fact that $\psi' = -\alpha_2(-\alpha_1\theta + \phi) + \phi'$ by algebraic simplification. We can define a corresponding notion of composition using Definition 5 where the constraint is that the output motion of one mechanism be made equal to the input motion of the other.

When two concrete mechanisms, (K_1, C_1) and (K_2, C_2), that implement relations $\mathcal{R}_1$ and $\mathcal{R}_2$ are composed, a rigid connection is formed between the output links of K_1 and the input links in K_2. The constraints, C_1 and C_2, are merged.

Definition 8 *Let the concrete mechanisms (K_1, C_1) with $K_1 = (V_1, E_1)$ and (K_2, C_2) with $K_2 = (V_2, E_2)$ implement $\mathcal{R}_1 \subseteq I_1 \times O_1$ and $\mathcal{R}_2 \subseteq I_2 \times O_2$ respectively. The new concrete mechanism (K, C) that implements $\mathcal{R}_1 \bowtie \mathcal{R}_2$ can be formed from (K_1, C_1) and (K_2, C_2), (denoted $(K, C) = (K_1, C_1) \bullet (K_2, C_2)$) by having $C = C_1 \cup C_2$, $V = V_1 \cup V_2$, $E = E_1 \cup E_2$ and by adding edges denoted rigid connections between output links in V_1 and corresponding input links in V_2.*

More complex compositions of multi-DOF abstract mechanisms and their concrete counterparts can be performed in our framework.

5 Formal Problem Specification

We now cast the synthesis problem in the formal framework for motions and mechanisms that we have just introduced. Synthesis problems are rarely posed with complete specifications of the desired input and output motions. Typically, constraints on the motions are given which define a class of abstract mechanisms, and not a particular one.

Definition 9 *The general synthesis problem is to find a concrete mechanism (K, C), given constraints on an abstract mechanism $\mathcal{R} \subseteq I \times O$.*

The construction of K from the partial specification of the relation $\mathcal{R}$ solves the *type* and *number* synthesis problem. The solution of C solves the *dimensional synthesis* problem.

Like [15], we view mechanisms as motion transformers. Our compositional approach to solving the general synthesis problem grounds the syntheses in a set of primitive motion relations (abstract mechanisms) $\mathcal{R}_p$ and associates with each a set of concrete implementations (K_p, C_p). In the first phase of synthesis, we find a composition of abstract mechanisms which satisfies the given constraints on $\mathcal{R}$. In the second phase, we choose concrete implementations of the primitive relations. We will show that this method is sound: that is, it produces concrete mechanisms which implement the given motion specifications.

Definition 10 *The abstract synthesis problem is to find a sequence of abstract mechanisms, $\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_n$, with known concrete counterparts, where $\mathcal{R}_1 \bowtie \mathcal{R}_2 \bowtie \ldots \bowtie \mathcal{R}_n$ satisfies the given constraints on $\mathcal{R}$.*

A simple incremental, generate-and-test algorithm for solving this problem starts with the identity abstract mechanism $\mathcal{I}$ $(I = O)$ and computes joins of primitive relations $\mathcal{R}_p$ until the composition satisfies the specification. For each primitive abstract mechanism we nondeterministically pick an implementation. We use Definition 8 to compose concrete mechanisms. This process involves simultaneously solving dimensional and geometric constraints from each of the chosen primitive implementations.

Definition 11 *The concrete synthesis problem is to find a set of concrete implementations (K_i, C_i), $1 \leq i \leq n$ for each element in the composition $\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_n$, which solves the abstract synthesis problem for the given constraints on $\mathcal{R}$, such that $\cup_{i=1}^{n} C_i$ is satisfiable.*

Theorem 1 The synthesis process: solving the abstract synthesis problem followed by concrete synthesis, is sound. That is, if it produces a solution, it will satisfy the specified motion constraints.

To prove completeness, we need to establish that the method will find an implementation of all specifications expressible in the configuration space formalism. The only way to prove this is to show that our primitive relation set can reconstruct any configuration space relation. Currently, our set of abstract mechanisms is incomplete: work is underway to construct a complete set.

We now discuss the computational complexity of the abstract and concrete synthesis problems. Let the cardinality of the set of primitive abstract mechanisms be n. Suppose we consider only composite mechanisms with at most p primitives. In the abstract synthesis phase, the generation component can explore $\sum_{i=1}^{p} n^i$ alternatives. We have to compute compositions during the process, which involves intersection of algebraic sets: the worst case time complexity is doubly exponential in the number of variables in the constraint set when a closed form solution is possible. In the concrete synthesis phase, the number of possible candidates are d^n where d is the maximum number of concrete instantiations for a primitive abstract mechanism. Each step in the concrete synthesis phase involves checking that a given non-deterministic choice of primitive implementations yields a consistent constraint set. The complexity of solving these geometric and dimensional constraints is the same as that of solving constraints generated in the abstract synthesis phase. Algebraic descriptions are extremely general, but suffer from two disadvantages. They require detailed knowledge of the configuration spaces and the computation of $\bowtie$ is very expensive in this representation. This motivates a qualitative approach to representing motions and the construction of the qualitative counterpart of the $\bowtie$ operator on configuration spaces. This is the subject of the next section.

6 Tractable Representations for Compositions

Qualitative descriptions partition the space of possible motions into equivalence classes. They have two chief advantages: they permit partial specification of motions. Second, they allow for potential efficiency gains in performing the join computations by eliminating the need to solve complex non-linear equations.

Our qualitative language is a predicate language that abstracts algebraic motion descriptions. It is similar to other motion languages in the literature [11, 14, 16, 24] in its use of predicate calculus. However, unlike these approaches, but in common with [13], our aim is to provide an analysis of tradeoffs between expressive power and computational efficiency for qualitative motion languages. We develop a soundness criterion called the ⋈-*preservation* property that a qualitative motion language must satisfy to generate correct syntheses.

A qualitative motion language can be characterized by a homomorphic mapping *A*, from a motion relation to a qualitative description, which picks out specific properties of a motion. For instance, our symbolic language represents rotations by their centers (xyz location), their axes (a unit vector), a speed (a constant for a uniform rotation), an angular range (for constrained rotations), and a frequency (for rotations that change sense). Rectilinear translations are represented by an axis (a unit vector), a speed, a range (for constrained translations), and a frequency (for reciprocations). How can we determine the representational adequacy of such a language for a given class of design tasks? For our synthesis task, we require the computation of compositions. If we can compute $A(\mathcal{R}_1 \bowtie \mathcal{R}_2)$ accurately and efficiently from $A(R_1)$ and $A(\mathcal{R}_2)$, for motion relations R_1 and R_2, we have an adequate language. The formal property is called ⋈-preservation and requires the specification of ⋈—the composition operation in the abstract language defined by A

Definition 12 *The mapping A preserves joinsitf*

$$A(\mathcal{R}_i \bowtie \mathcal{R}_j) = A(\mathcal{R}_i) \bowtie A(\mathcal{R}_j)$$

Note that this constraint places restrictions on the definition of ⋈. We illustrate the ⋈ computation using the gear train example of Section 4. G_1 and G_2 transform rotary motion to another rotary motion with a different speed and sense. The angular velocity a, can be obtained from the algebraic description of motion: $\psi(t) = -\alpha\theta(t) + \phi$, where $\psi(t)$ is the output configuration at instant *t*, and $\theta(t)$ is the configuration of the input link at *t*. The predicates below reformulate the algebraic descriptions of rotation provided earlier.

$$\mathcal{G}_1 = \begin{array}{ll} \textbf{Input}\text{: rotation} & \textbf{Output}\text{: rotation} \\ \texttt{center}\text{: } (x_A, y_A) & \texttt{center}\text{: } (x_B, y_B) \\ \texttt{velocity}\text{: } S_1 & \texttt{velocity}\text{: } -\alpha_1 S_1 \end{array}$$

$$\mathcal{G}_2 = \begin{array}{ll} \textbf{Input}\text{: rotation} & \textbf{Output}\text{: rotation} \\ \texttt{center}\text{: } (x_M, y_M) & \texttt{center}\text{: } (x_N, y_N) \\ \texttt{velocity}\text{: } S_2 & \texttt{velocity}\text{: } -\alpha_2 S_2 \end{array}$$

$$\mathcal{G}_1 \bowtie \mathcal{G}_2 = \begin{array}{ll} \textbf{Input}\text{: rotation} & \textbf{Output}\text{: rotation} \\ \texttt{center}\text{: } (x_A, y_A) & \texttt{center}\text{: } (x_F, y_F) \\ \texttt{velocity}\text{: } S_1 & \texttt{velocity}\text{: } -\alpha_2(-\alpha_1 S_1) \end{array}$$

To implement ⋈ by ⋈, we equate the description of the output motion of $\mathcal{G}_1$ with that of the input motion of $\mathcal{G}_2$, where $\mathcal{G}_2$ may be rigidly transformed. The composition results in the following constraints.

1. $T_{rigid}(x_M, y_M) = (x_B, y_B)$
2. $T_{rigid}(x_N, y_N) = (x_F, y_F)$

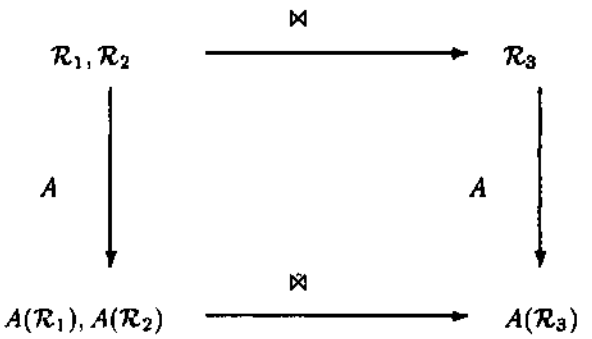

Figure 2: Preservation of joins by abstraction *A*

We calculate the rigid transformation T_{rigid} (in this case a pure translation) which moves the rotation about the *z* axis from (x_M, y_M) to (x_B, y_B) and we then apply that transformation to the output of $\mathcal{G}_2$.

In general, for the predicate language of rotations and rectilinear translations introduced above, ⋈ "unifies" motions M_i and M_j by calculating the generalized rigid transformation T such that $TM_i = M_j$. There are specific conditions under which two motions can be unified: they have to be of the same "type". Two rotations can be unified if they have the same axes of rotation, their angular speeds and range are equal, and they have the same frequency (if they are oscillations). The unification process captures the join calculation in C-space, and the abstract description filters exactly the properties of interest.

To demonstrate the computational efficiency gained by the loss of expressive power, consider the slider crank mechanism with the crank at (0, 0, 0) and the slider that moves along the *x* axis, with crank radius *r*, and with *l* being the length of the link connected to the slider. The abstract motion relation implemented by the slider crank is:

$$\mathcal{S} = \{(0,0,\theta,x,0,0) \mid x = r\cos\theta + l\left[1 - \sqrt{1 - \left(\tfrac{r}{l}\right)^2 \sin^2\theta}\right]\} \quad (2)$$

$$\mathcal{S}_q = \begin{array}{ll} \textbf{Input}\text{: rotation} & \textbf{Output}\text{: translation} \\ \texttt{center}\text{: } (0,0) & \texttt{interval}\text{: } (r-l, r+l) \\ \texttt{velocity}\text{: } S & \texttt{frequency}\text{:} S \end{array}$$

Our qualitative description only picks out the motion interval and frequency of the slider. The motion interval is calculated from Equation 2 by using the fact that $\theta \in [0, 2\pi)$. The angular velocity can be derived by examining the algebraic description of the slider's motion as a function of time. These two aspects of the slider's motion are sufficient characterizations for the ⋈ calculation. We no longer need to solve the non-linear equation 2.

If the configuration space relations $\mathcal{R}_1$ and $\mathcal{R}_2$ are linear (representable by linear constraints), the ⋈ operation defined above permits *A* to be join-preserving. To better understand join preservation, it is useful to consider the example in Figure 3 where it is violated. The output

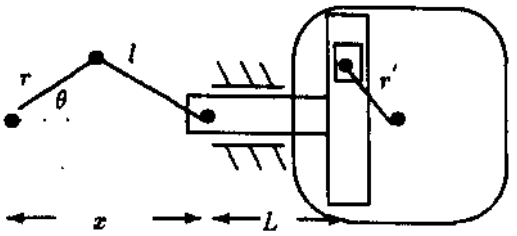

Figure 3: An example that shows join preservation violation

of the crank rocker in Figure 3 is a rectilinear translation (reciprocation) which is the input to the skotch-yoke mechanism. Our ⋈ construction unifies the two motions as long as their ranges, speed, and axes coincide. Unfortunately, an analysis of the underlying C-space[3] relations reveals that there is at most one possible position where the two motions would intersect, thus the mechanism would jam (become rigid). Both the crank-rocker and skotch-yoke mechanisms are non-linear.

The specific ⋈ developed above is inadequate for handling non-linear mechanisms in a general way. To guarantee preservation of joins for the *specific* ⋈ and *A* combination introduced here, we ensure that non-uniform motions are not composed. This can be done with the restriction that non-linear mechanisms only take uniform motions as input.

7 Efficient Synthesis Algorithms

We reformulate the algebraic description of the abstract synthesis problem in terms of qualitative motion descriptions.

Given i, a qualitative specification of the input motion: o, a qualitative specification of the output motion, and constraints on *i* and *o.*

Find a sequence of abstract mechanisms $A_1,...,A_n$ which when composed will transform any motion described by *i* to some motion described by *o.* To be exact, we want

$$\forall m_i \in i. \exists m_o \in o. (m_i, m_o) \in \mathcal{A}_1 \bowtie \ldots \bowtie \mathcal{A}_n$$

where m_i E *i* means that the motion m_i, is in the class of motions described by the qualitative specification i.

To make the process efficient, we transform the naive generate-and-test scheme to a goal-directed procedure in Section 4 that chains backward from the desired output *o* to i. We distinguish between single-input, single-output (SISO) mechanism synthesis from single-input, multi-output mechanism (SIMO) synthesis because of the opportunity for optimization by function sharing in the latter case. We begin with the algorithm for the SISO case.

The algorithms below are *not committed to any particular abstraction language.* For each language, we require procedures that test equality of motion descriptions, and regress constraints on motion through a primitive abstract mechanism. We will illustrate these in the context of the simple motion description system introduced in the previous section.

7.1 Synthesizing single input, single output mechanisms

The recursive algorithm for synthesizing single input, single output mechanisms is shown in Table 1. We imple-

SISO_Synthesize(i,o)

1. **If $i = o$, then return the null machine.**
2. **Else select an abstract mechanism $\mathcal{M}$ and find a rigid transformation T such that**

$$q = \{m \mid (m, m_o) \in T(\mathcal{M}), m_o \in o\} \neq \emptyset$$

   Thus *q* describes the (largest) set of motions that can be transformed by *T(M)* to motions in *o. q* is the most general qualitative description in the motion language which meets that the previous requirement. It is the regression or backprojection of *o* with respect to *T(M).*

3. Return *[T{M),* SISO-Synthesize(i,q)]

Table 1: Algorithm for synthesizing single input, single output mechanisms

ment the synthesis method for our language as a depth-bounded, goal-directed, depth-first backward chainer in CLP(R). The operational model of CLP(R) is similar to Prolog (so the reader familiar with Prolog can read the code below quite easily), however unification is replaced by a more general mechanism: solving constraints in the domain of functors over real arithmetic terms.

For a linear primitive mechanism, we store its name, a scaling factor for the input and output motions, and the types of input and output motion. For example, the abstract mechanism corresponding to a gear pair with gears of sizes 3 and 5 is represented as:

```
mechanisrn(gear-pair,
    linear(-3/5),
    [rotation((0,0,0),(0,0,l))],
    [rotation(((3+5)/2,0,0),(0,0,l))]).
```

For a non-linear primitive mechanism, we store its name, its input motion which must be uniform, and its output motion. For instance, the abstract non-linear mechanism corresponding to a crank rocker is represented as:

```
mechanism(crank-rocker( L),
    non-linear,
    [rotation((0,0,0),(1,0,0)), velocity(F)],
    [rotation((0,L,0),(1,0,0)), frequency(F), range(R)])
    :- R > 0, R < π .
```

The top-level invocation of the synthesis function is: synthesize(input motion, output motion, null design, depth bound). The base case of the synthesis occurs when the input motion *i* is equal to the output motion

[3] $K - \sin\phi = r\cos\theta + l[1 - \sqrt{1 - (\frac{r}{l})^2 \sin^2\theta}$

o: this is established by solving arithmetic constraints that are generated when the motions are unified.

```
synthesize(In_motion, Out.motion, Design, Depth) :-
    motion_eq(In_motion, Out_motion).
```

motion_eq is a predicate testing whether two motion descriptions are equivalent. Its actual implementation depends on the specific motion language.

The recursive step of synthesis first involves nondeterministic choice of a primitive mechanism. Suppose a linear primitive mechanism with output motion o_p is chosen. The output o_p is made equal to the output of the overall mechanism via a rigid transformation T computed by solving the constraint $T(o_p) = o$. Then, the new synthesis problem $i, T(i_p)$ is solved, where $T(i_p)$ is the input motion of the primitive after rigid transformation. We can calculate $T(i_p)$ very simply from the type of input motion and the scaling factor that relates the input and output motions of the mechanism.

```
synthesize(In_motion, Out.motion,
    [(N,R_transform)|Design], Depth) :-
    Depth > 0,
    mechanism(N, linear(F), PJnjnotion, P_out_motion),
    transform(P_out_motion, Out.motion, R_transform),
    linear -apply (R_.transform, F, P_in_motion, NewGoal),
    synthesize(In_motion, NewGoal, Design, Depth-1).
```

If, however, the primitive mechanism chosen is not linear, the the input and output motion description, i_p and o_p, have to be provided explicitly. We compute the rigid transformation T by solving the matrix equation $To_p = o$.

```
synthesize(In_motion, Out.motion,
    [(N,R_transform)|Design], Depth) :-
    Depth > 0,
    mechanism(N, nl, P_In_motion, P.outjnotion),
    transform(P_out_motion, Out_motion, R.transform),
    nl_apply(R.transform, P_in_motion, NewGoal),
    synthesize(In_motion, NewGoal, Design, Depth-1).
```

Consider the problem of designing a windshield wiper introduced in Section 1. The input power is provided by a motor rapidly rotating around the *z* axis. The wiper oscillates in the *yz* plane with low frequency. We specify the problem as follows. Capitalized symbols in the descriptions above are variables. The constraints specify ranges on some of the variables.

Input motion i	**Output motion** o
rotation	rotation
center: (0,0,0)	center: $(0,Y,Z)$
axis: (0,0,1)	axis: (1,0,0)
velocity: 20	frequency: F, range: X
Constraints: $0.5 < F < 2$, $\pi/2 < X < \pi$	

For the synthesis of the wiper, the first primitive abstract mechanism chosen is a crank rocker.

Input motion	**Output motion**
rotation	rotation
center: (0,0,0)	center: $(0,L,0)$
axis: (0,0,1)	axis: (0,0,1)
angular_velocity: F	frequency: F, range: R
Constraints: $0 < R < \pi$	

To have the output of this primitive mechanism be a rotation at $(0, Y, Z)$, we will need to move the crank rocker. We have to solve two systems of equations to find the rigid transformation. The first system arises from the requirement that the axes of rotations have to be parallel. We need to find the angles of rotation that allign the axes. We have to solve

$$R\begin{pmatrix}0\\0\\1\end{pmatrix} = \begin{pmatrix}1\\0\\0\end{pmatrix}$$

where R is a 3x3 rotational matrix. Let ψ, ϕ, θ be rotational angles about the z, x, and y axis respectively.

The second system of equations arises because we require the centers of rotations to be coincident after the rigid transformation. Let T be the 4x4 transformation matrix.

$$T\begin{pmatrix}L\\0\\0\\1\end{pmatrix} = \begin{pmatrix} & R & & x\\ & & & y\\ & & & z\\ 0 & 0 & 0 & 1\end{pmatrix}\begin{pmatrix}L\\0\\0\\1\end{pmatrix} = \begin{pmatrix}0\\Y\\Z\\1\end{pmatrix}$$

From these equations, we solve for $x, y, z, \theta, \phi, \psi$. One solution to this problem produced by our system is

$$x = 0\ y = Y - L,\ z = 0,\ \theta = \pi/2,\ \phi = 0, \psi = 0.$$

The rigid transformation computed in this case is a rotation of $\pi/2$ about the y axis and a translation of $Y - L$ along the y axis. We apply the transformation to the input motion of the primitive mechanism. The intermediate motion generated is:

Intermediate motion q
rotation
center: $(0, Y - L, Z)$
axis: (1,0,0)
angular_velocity: F

All that remains is to transform the intermediate motion generated to the specified input motion i. This is accomplished with another abstract mechanism that meets the constraint on F and changes the axis of rotation, in this case, a worm-spur pair.

Input motion	**Output motion**
rotation	rotation
center: (0,0,0)	center: $(0,W,0)$
axis: (0,0,1)	axis: (1,0,0)
angular_velocity: F_1	angular_velocity: F_1/α
Constraints: none	

The unification of the intermediate motion generated above with the output of this primitive abstract mechanism generates the constraints: $F = F_1/\alpha$, $Y - L = W$, $Z = 0$. The rigid transformation is the identity transformation. Now the input motion of this primitive mechanism can be matched with the input description i yielding $F_1 = 20$. This generates the derived constraint $F = 20/\alpha$. The synthesis and refinement process generates the constraints on the configuration of both mechanisms, the dependency of X, Y and Z on the dimensions of the links (W, L), as well as the fact that $F = 20/\alpha$, where α is the worm-spur gear ratio. Alternate abstract

designs and the corresponding refinements of a windshield wiper found by our system include the composition of a worm-spur pair, a slider crank, and a rack and pinion mechanism; as well as a worm-spur, scotch-yoke and a rack-and-pinion mechanism. This example shows how the backward chaining process accumulates simple algebraic constraints which are solved incrementally during the synthesis. The constraint programming language CLP(R) [27] is used to implement the algorithm. CLP(R)Graphical outputs are produced via an interface to Mathematica[4]

Theorem 2 Algorithm SISO_Synthesize (i,o)) is sound: i.e, it designs concrete mechanisms that satisfy the qualitative motion specifications (Q_i, Q_0)-

This theorem can be proven by induction on the length of the generated solution. The worst case complexity of this algorithm is exponential in the length of the solution produced. The worst case branching factor for the search is around 20, corresponding to the number of primitive motion relations. In practice, the average branching factor is much smaller (around 2 for the examples in this paper) because the constraint accumulation process is a least-commitment strategy that minimizes backtracking in the space of compositions of primitive abstract mechanisms. In other words, we incrementally solve for the rigid transformation and dimensions of primitives during synthesis. We do not search for them discretely, which may be very time consuming. Put another way, our constraint-based representation allow us to perform delayed instantiation of parameters. Each search path encodes a whole class of solutions. Pruning or accepting a path involves pruning or accepting a whole class of solutions. The algorithm synthesizes many of the designs for conversion of uniform rotation to reciprocation in [1] in a few seconds. The synthesis of the wiper shown in Figure 1 and its variants was also completed in about two seconds on a Sparc 1 +

7.2 Synthesizing single input, multiple output mechanisms

Many useful mechanisms produce multiple outputs from a single source, e.g., eggbeaters, cars. To design these single input multiple output (SIMO) mechanisms, we need to specify a sequence of output motions.

Given i, a qualitative specification of the input; $o^1, \ldots, o^n$, a sequence of output motions, and constraints on i and o's.

Find A tree of abstract mechanisms, $\mathcal{A}^1_1, \ldots, \mathcal{A}^1_{k_1}, \mathcal{A}^2_1, \ldots, \mathcal{A}^n_{k_n}$, which when composed satisfies the input-output specification.

The SIMO synthesis problem can be solved by a series of calls to SISO_Synthesize as in (Table 2). Calls to SISO-Synthesize produce a tree with isolated paths from i to each o^j. However, this introduces a lot of redundancy in the form of common intermediate motions along these paths. The optimization algorithm in Table 2 merges common motions in the paths: if

[4]Mathematica is a trademark of Wolfram Research, Inc.

SIMO_Synthesize$(i, o^1, \ldots, o^n)$
for j from 1 to n do
  $\mathcal{A}^j_1, \ldots, \mathcal{A}^j_{k_j} \leftarrow$ **SISO_Synthesize**(i, o^j)
/* optimization */
for j from 1 to n do
  for each $\mathcal{A}^j_k$ in $\mathcal{A}^j_1, \ldots, \mathcal{M}^j_{k_j}$ do
    for l from j+1 to n do
      for each $\mathcal{A}^l_m$ in $\mathcal{M}^l_1, \ldots, \mathcal{A}^l_{k_l}$ do
        if **InMotion**$(\mathcal{A}^l_m) =$ **InMotion**$(\mathcal{A}^j_k)$ then
          delete $(\mathcal{A}^l_1, \ldots, \mathcal{M}^l_{m-1})$
          merge the inputs of $\mathcal{A}^l_m$ and $\mathcal{M}^j_k$

Table 2: Algorithm for Synthesizing single input, multiple output mechanisms

the inputs I_1 and I_2 to two abstract mechanisms in two different branches of the initial tree are equivalent, we eliminate the path to I_2 and connect I_1 in place of I_2. This eliminates repeated transformation of i to I_2. In the algorithm, we denote primitive $\mathcal{A}$'s input motion by **InMotion**$(\mathcal{A})$. When we merge the inputs of $\mathcal{A}^l_m$ and $\mathcal{A}^j_k$ in the last step of **SIMO_Synthesize**, the input links of $\mathcal{A}^l_m$ and $\mathcal{A}^j_k$, as well as the output links of $\mathcal{A}^j_{k-1}$ are rigidly connected together. The optimization step merges identical motions, and therefore the final mechanism produced by the synthesis is behaviorally equivalent to the original tree. By the correctness of **SISO_Synthesize** we can conclude that **SIMO_Synthesize** is also sound. The optimization reduces the number of nodes in the tree and thus produces more compact refinements. Thus we have the following theorem.

Theorem 3 *Algorithm* **SIMO_Synthesize**$(\mathcal{I})$ *is sound: i.e, it designs concrete mechanisms that satisfy the motion specifications* $(Q_i, (Q^1_o, \ldots, Q^n_o))$.

We now present the class of mechanisms that are synthesizable by these algorithms. Clearly the class is determined by the qualitative motion description language used, and the set of primitive abstract mechanisms and their associated implementations. For the specific motion language used in our current implementation, the class of mechanisms synthesizable are fixed-topology, single-degree of freedom mechanisms with at most one nonlinear mechanism on each path from the input to the outputs. The mechanisms we consider thus far are composed of rigid parts. The single-degree-of-freedom restriction applies in our case, because all of our primitive motion relations have only one degree of freedom. The composition of two or more mechanisms with single degree of freedom can only produce mechanisms with at most one degree of freedom. Multi-degree of freedom mechanisms can be synthesized by our algebraic technique. The restriction on rigid parts obtains because our definitions of motions and mechanisms are grounded in configuration spaces of rigid bodies. By allowing definitions based on generalized configuration spaces, we can allow for some limited forms of non-rigidity. The restriction on the number of non-linear mechanisms in

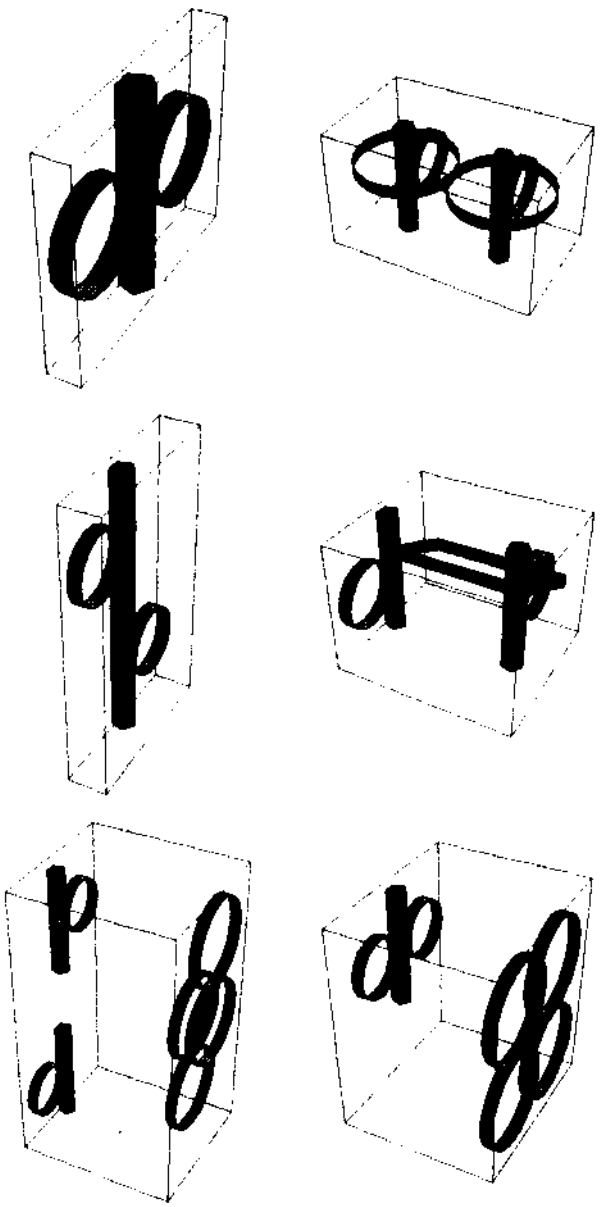

Figure 4: Conceptual Designs for an Eggbeater

a design is needed for the correctness of the abstraction *A* that generates the qualitative motion language. The fixed-topology restriction can be eliminated by having a richer set of primitive relations as well as a richer motion specification language which allows for expression of when and how part contacts are made and broken. A limitation of our current approach is the lack of a component for shape design. If there is no sequence of primitive relations that satisfies the given specification, our method will fail to produce a design. We can integrate the methods of [11] for synthesizing novel shapes into our design system to automatically extend our library of primitive abstract mechanisms.

8 Conclusions

This paper presented a case study of the integration of methods in qualitative physics and constraint programming with general algebraic reasoning with configuration spaces. The design domain studied is that of kinematic synthesis of mechanisms from specifications of input and output motions. Two algorithms were presented that rapidly generate alternate behavioural decompositions and concrete refinements of a mechanism. We also identified the class of mechanisms which can be correctly synthesized within the qualitative framework. We have implemented our method in CLP(R) and all examples discussed in this paper are drawn from our implementation. Our base set of examples are drawn from mechanisms in [19] and [1]. We are presently enriching the language of qualitative motion specifications to handle richer classes of non-linear motions. This will allow us to obtain better coverage over the examples in the compendia listed above. Future work involves extending the set of primitive relations, proving completeness properties for these relations, and integrating mechanism synthesis with multi-domain (including dynamics and optics) designs.

There are other approaches to mechanism synthesis that can be profitably combined with the first-principles approaches discussed above. Expert system techniques [26] for synthesizing special classes of mechanisms e.g., cam-follower mechanisms, occupy an interesting middle ground between p re-parametric design schemes which requires high-level qualitative specifications and the numerical optimization packages which require very detailed kinematic specifications. Case-based methods [3, 20] for synthesis of mechanical systems begin with a known library of designs and use the goal specification to index relevant designs. The retrieved designs are modified to meet the given specifications. The algorithm developed here can be used to design indices for the library of designs. This works by running the synthesis algorithm "in reverse" to parse or understand a design in terms of given primitive motion relations.

The class of conceptual design tasks that can profit from the integration we have effected are tasks with a significant geometric component. We have developed fast simulation methods for the class of mechanisms that can be synthesized by the algorithms presented here. Space limitations preclude their inclusion in this document; a discussion of simulation methods will be present

in the talk. All the physical prototyping of the designs presented in this paper were performed using Technics Lego. Integrating conceptual design systems through detailed design and physical prototyping in a standard medium, will be discussed in the talk. The talk will focus on mechanical nano-technology designs because low-dimensional configuration spaces can be used to reason about shapes and motions in that domain. Computational scale issues and our experience with the field testing of our tools at Xerox will also be presented.

References

[1] I. Artobolevsky. Mechanisms in Modern Engineering Design, vols. 1-4- MIR Publishers, Moscow, 1979. English translation.

[2] J. Cagan and A. Agogino. Innovative design of mechanical structures from first principles. Journal of Artificial Intelligence in Engineering Design and Manufacturing, 1(3): 169-189, 1987.

[3] K.P. Sycara D. Navinchandra and S. Narasimhan. A transformational approach to case-based synthe sis. Journal of Artificial Intelligence in Engineering Design and Manufacturing, 5(2), 1991.

[4] B. Faltings. Qualitative kinematics in mechanisms. Artificial Intelligence, 44:89 119, 1990.

[5] S. Finger and J. Rinderle. A transformational approach to mechanical design using a bond graph grammar. In Proceedings of the 1st ASME Design Theory and Methodology Conference, Montreal, Canada, 1989.

[6] F. Freudenstein and L. Dobrjanskyj. On a theory of type synthesis of mechanisms. ASME Transactions on Machines and Mechanisms ??, 196?

[7] F. Freudenstein and Maki. The creation of mechanisms according to kinematic structure and function. Environment and Planning B, 6:375-391, 1979.

[8] F. Freudenstein and L.S. Woo. Kinematic structure of mechanisms. In W.R. Spillers, editor, Basic Questions of Design Theory. North Holland, 1974.

[9] J. Rinderle and S. Finger. A synthesis strategy for mechanical devices. Research in Engineering Design, 1:(1), 1989.

[10] G. Iyengar, C. Lee and S. Kota. Towards an objective evaluation of alternative designs. In Proc. Design Theory and Methodology, ASME 1992.

[11] L. Joskowicz and S. Addanki. From kinematics to shape: An approach to innovative design. In Proceedings of AAAf-88, pages 347-352. Morgan Kaufmann, 1974.

[12] L. Joskowicz and E. Sacks. Computational kinematics. Artificial Intelligence, 51:381 416, 1991.

[13] L. Joskowicz. Mechanism Comparison and Classification for Design. In Artificial Intelligence in Engineering Design, Vol II, ed. C. Tong and D. Sriram, Academic Press, 1993.

[14] S. Kannapan and K. Marshek. Design synthetic reasoning. Technical Report 216, Mechanical Systems and Design, University of Texas at Austin, September 1989.

[15] S. Kota. A qualitative matrix repreentation scheme for the conceptual design of mechanisms. In Proceedings of the ASME Design Automation Conference. ASME, 1990.

[16] S. Kota. Qualitative motion synthesis: Toward automating mechanical systems configuration. In Proceedings of the NSF Design and Manufacturing Systems Conference, pages 77—91, 1990.

[17] J.C. Latombe. Robot Motion Planning. Kluwer Academic Publishers, 1991.

[18] H. M. Mabie and C. F. Reinholtz. Mechanisms and Dynamics of Machinery, 4th edition. John Wiley and Sons, 1987.

[19] D. Macaulay. The Way Things Work. Houghton Mifflin Company, 1988.

[20] K.P. Sycara R. Guttal J. Koning S. Narasimhan D. Navinchandra. Cadet: A case-based synthesis tool for engineering design. International Journal of Expert Systems, 1991.

[21] G. Pahl and W. Bietz. Engineering Design. The Design Council, Springer-Verlag, 1984.

[22] H.M. Paynter. Analysis and Design of Engineering Systems'MIT Press, 1961.

[23] M M. Rueleaux. The Kinematics of Machinery. MacMillan & Co., 1876. Translated by Alex B.W. Kennedy.

[24] K. Ulrich. Computation and pre-parametric design. Technical Report 1043, MIT Artificial Intelligence Laboratory, July 1988.

[25] K. Ulrich and W. Seering. Conceptual design: Synthesis of systems of components. In S. Chandrasekar C.R. Liu, A. Requicha, editor, Intelligent and Integrated Manufacturing Analysis and Synthesis. ASME, PED-Vol 25, 1988.

[26] B. Yang U. Datta P. Datseris Y. Wu. An integrated system for design of mechanisms by an expert system: Domes. AI EDAM, 3(1):53 70, 1989.

[27] N. Heintze S. Michaylov P. Stuckey R. Yap. The CLP(R) programmer's manual, version 1.1. Technical report, Carnegie-Mellon University and IBM Research, 1991.