Kinematic Synthesis with Configuration Spaces*

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

Cheuk-San (Edward) Wang
Department of Computer Science
Cornell University
Ithaca, NY 14853
wang@cs.cornell.edu

April 10, 1992

Abstract

This paper introduces a new approach to the conceptual design of mechanical systems from qualitative specifications of behavior. The power of the approach stems from the integration of techniques in qualitative physics and constraint programming. We illustrate the approach with an effective kinematic synthesis method that reasons with qualitative representations of configuration spaces using constraint programming.

1 Introduction

The overall 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, and with the need to build in continuous improvement into designs, 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 qualitative physics and constraint programming to build new computational prototyping tools for conceptual design.

This paper presents a conceptual design system in the area of mechanism synthesis. Mechanisms are an important part of most electro-mechanical systems — they form the basic geometric elements

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

of such systems. Mechanisms transmit motion from one rigid body to another. An example is the mechanism in a quartz watch that uses the oscillating crystal to drive the minute hand at sixty times the rotational speed of the hour hand. Another is a rack and pinion steering of an automobile that converts rotation about one axis (the steering action) to reciprocating motion along another axis. 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. It 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 specialized expert systems or solvers using cost, material, manufacturing and assembly constraints.

We illustrate how our design system is used with a simple example. Consider 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. We can describe these as a set of constraints in our predicate motion vocabulary as follows. Rotations are specified by their centers and directions of axes: rotation(Center, Axis). As we shall see later in the paper, our motion predicates are a shorthand notation for describing sets of configuration spaces.

Input Motion: rotation((0,0,0),(0,0,1)), speed(20)

Output Motion: rotation((0,Y,Z),(1,0,0)), frequency(F), range(R),
0.5 < F < 2, $\pi/2 < R < \pi$

The first structure enumerated by the system is shown in Figure 1. This design solves the problem with $F = 0.833333, Y = 2, Z = -6$. It employs a worm-spur pair (with scaling factor 1/24) which converts the uniform input rotation around the z axis to one about the x axis. The spur gear drives a crank rocker of length 6. The wiper output is tapped from the rocker. Our program calculates the position and orientations of the gears and the crank rocker. Note that both the topology and the types of kinematic pairs of the mechanism are determined together with their actual dimensions. The physical implementation of the design in $Technics^{TM}$ Lego is in Figure 4. Another design, shown in Figure 2, that satisfies the same motion specifications, realizes the oscillatory output motion using a rack and gear pair, where the rack is driven by a slider crank. The crank is rotated uniformly by a worm-spur pair, where the worm is driven by the motor. This design is found in [23] (see Figure 3).

There are several unique aspects of our method. We have a uniform representation for a variety

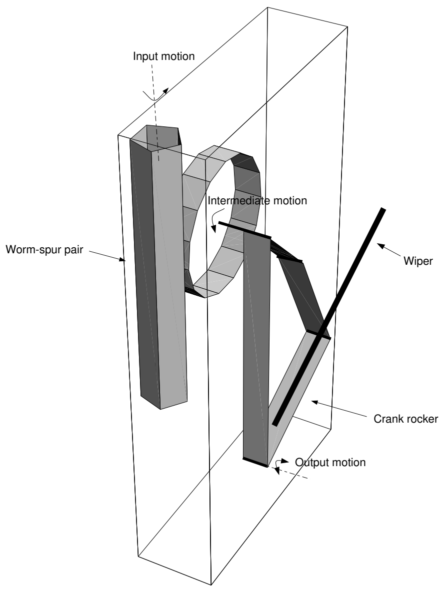


Figure 1: Design of a windshield wiper

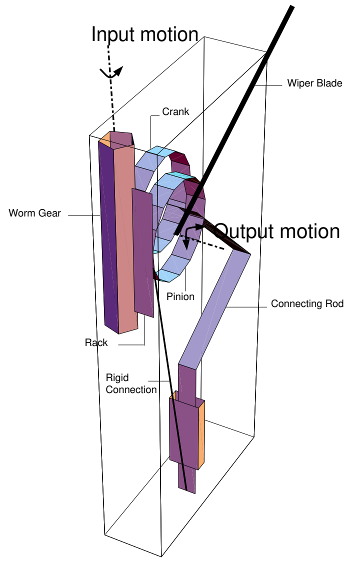


Figure 2: Another design of a windshield wiper

of constraints and can take them all 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. 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 three windshield wiper designs were generated and rendered in about a second each on a Sparcstation 1+.

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 $\bowtie$-*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. All the examples presented in this paper are designed by our system. Our design system enumerates designs ordered by increasing complexity[1]. Our system has produced innovative designs for a number of common devices described in [23].

1.1 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. The solution to the kinematic synthesis problem is structured in two phases: *type and number* synthesis and *dimensional* synthesis [22]. In type and number synthesis, the overall shapes of the kinematic links in the mechanism, their topological arrangement, and the nature of the kinematic connections between links, are determined. The link lengths are derived during dimensional synthesis. Sophisticated numerical packages are available for dimensional synthesis or detailed design [22, pg. 618], and research is now active [2, 6, 13, 16, 19, 28, 29, 30, 31] in the area of type and number synthesis or conceptual design.

Conceptual synthesis is generally acknowledged to be a very difficult problem. A modern textbook in the area [22] states that

> The designer generally relies on intuition and experience as a guide to type and number

[1] We have a simple metric for complexity which counts the number of primitives in a design.

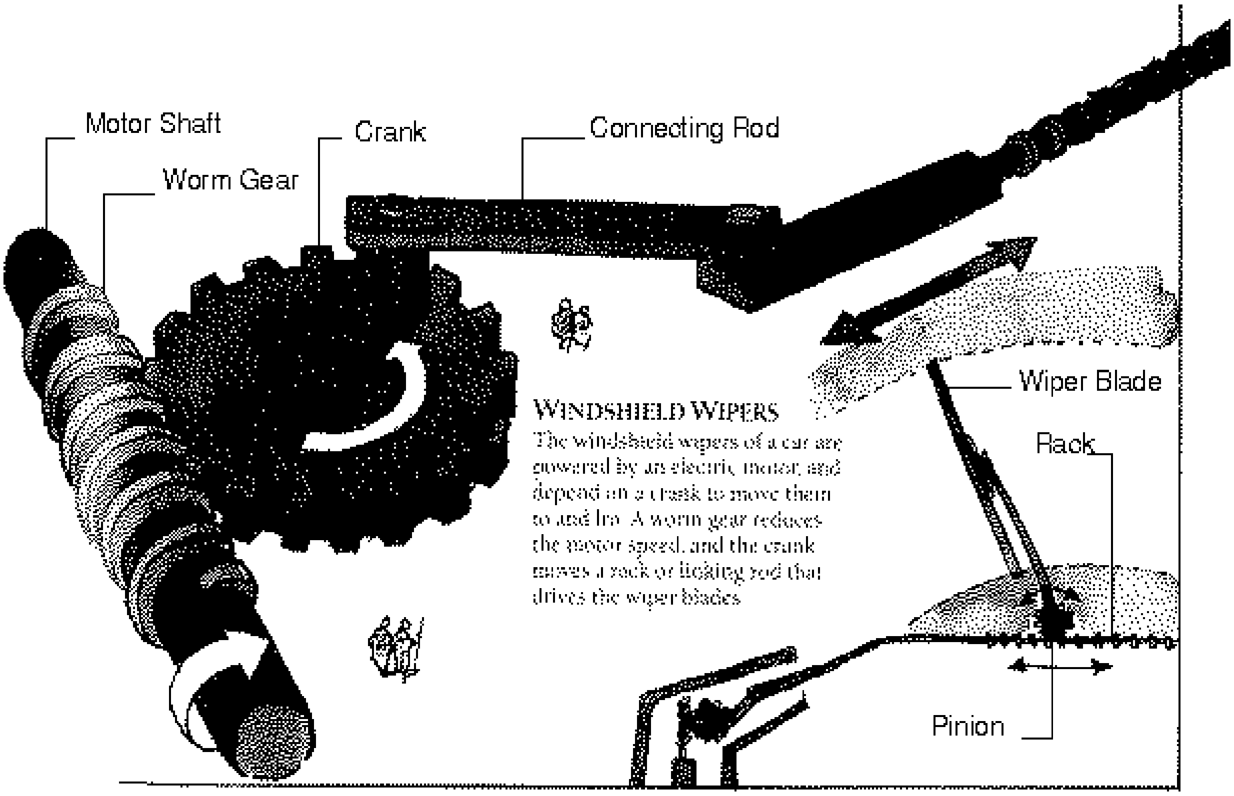


Figure 3: Windshield wiper design from Macaulay's *How Things Work*

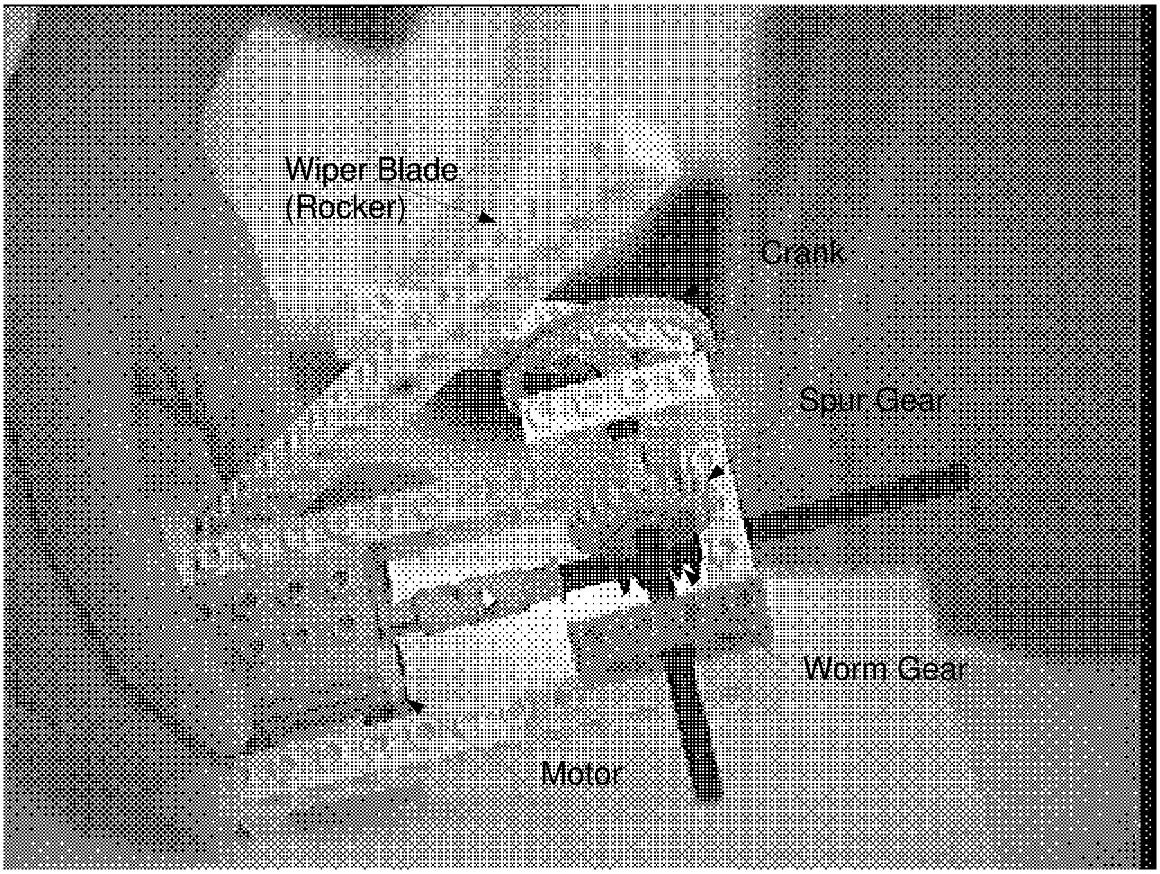


Figure 4: A Technics Lego implementation of the windshield wiper design in Figure 1

synthesis. Very little supporting theory is available in these areas.

The conceptual synthesis problem 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).[2] 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, 13, 16, 18, 19, 31]. Our paper extends the work in this area by providing a foundation for building compositional motion synthesis algorithms, and by designing and implementing a new, practical, constraint-based motion synthesis method.

One of the chief issues in the area of mechanism synthesis from specification of motion is the choice of motion description language. The choice of language is critical, because it determines the space of specifiable as well as constructible designs. A very fine-grained, quantitative description language allows us to make fine distinctions among motions. A very coarse qualitative language, on the other hand, limits expressiveness but allows for the construction of very efficient design algorithms. Our approach allows us to exploit the best features of quantitative and qualitative motion description languages. At the finest level of detail, motion is formalized in terms of *configuration spaces* [21]. The motion of an object is viewed as a mapping from time to its configuration space. We also develop a predicate vocabulary [15, 16, 18] that is an effective shorthand for some commonly occurring motions. Essentially, we define a type hierarchy of motion descriptions whose semantics are grounded in configuration spaces. The predicate language permits high-level reason-

[2] There is considerable variation in the use of the terms function and behavior in the design synthesis literature. We mean function to be the intended use of a device; that is, how it interfaces with the world. The function of a windshield wiper is to remove dirt from the surface of a car's windshield. Behavior, in the context of mechanisms, is the intrinsic motion characteristic of the device. A specification of the windshield wiper's behavior is provided in the first section of this paper. There are several behaviors that can achieve the windshield wiper function. Similarly, a given behavior can be used to achieve several functions. The wiper motion can be used to clean toner from a film in a copier. What makes mechanical engineering design interesting is the many-to-many mapping between the functional and behavioral specifications, as well as from behavioral specifications to structure.

ing with certain classes of motions and the design of efficient synthesis procedures. We specify the class of synthesis problems for which the abstraction sanctioned by the predicate descriptions yield correct designs in a computationally effective manner.

Our solution to the motion synthesis problem follows the standard breakdown into the conceptual and detailed phases. A mechanism is specified by its input-output motion relation. Conceptual synthesis is a compositional [19] process grounded in primitive motion relations. Each primitive motion relation has associated with it a set of concrete implementations. We first construct a schematic called an *abstract mechanism* that is a decomposition of the user-specified input-output motion specification in terms of the known primitive motion relations. Alternate decompositions yield alternate conceptual designs. Abstract mechanisms are refined by choosing implementations of the component primitive motion relations. Multiple refinements of a given abstract mechanism can be explored in this phase.

A concrete mechanism specification includes the topological arrangement of links and kinematic pairs that realize the input-output relation, as well as dimensional constraints on the links. To generate a concrete mechanism for a given specification, we compose the implementations of the primitive relations. This process accumulates constraints on link dimensions that are solved to obtain a fully instantiated design. One of the strengths of our approach is the clean integration between the phases of conceptual and detailed design: dimensional constraints generated during the conceptual phase are solved during the synthesis of concrete mechanisms.

1.2 Related Work

We present a summary of relevant related work and place our paper in context. Work on dimensional synthesis of mechanisms has a long history in the field of mechanical engineering and there are several packages for numerical optimization [22, Chapter 11]. Recently, Hoeltzel and Cheng [10] have developed methods for dimensional synthesis of four-bar linkages from specifications of planar trajectories by using neural nets to perform pattern matching over these trajectories. Work on type and number synthesis of mechanisms is an active research area and we classify the approaches into three main categories: structural, behavioral and functional (see Figure 5).

1. *Structural Theories*: These are theories that generate mechanism topologies systematically, usually from specification of number of links and total number of degrees-of-freedom (DOF) of the mechanism. Pioneering work in developing structural theories of type and number syn-

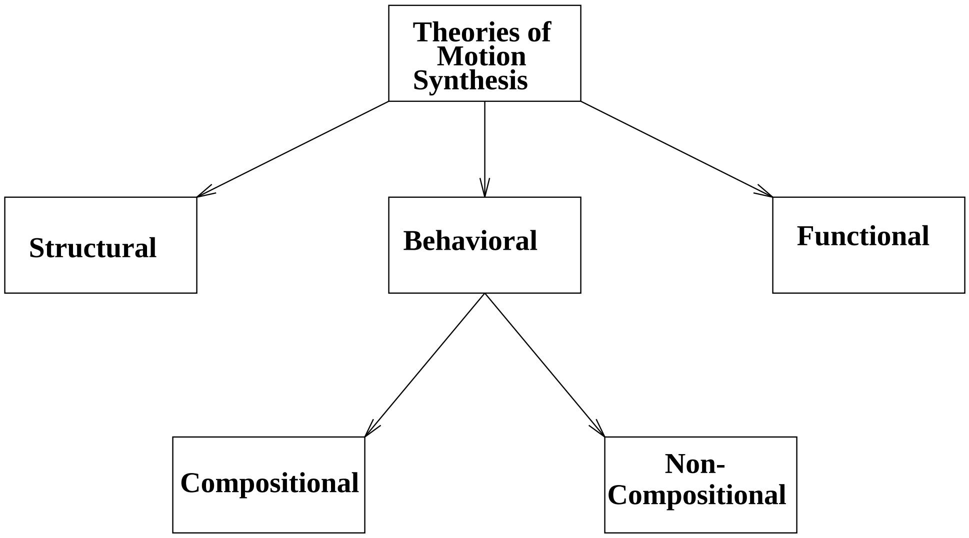


Figure 5: Theories of Motion Synthesis

thesis was undertaken by Freudenstein [6]. The key idea in Freudenstein's work is to represent mechanism topology using graphs. He presents combinatorial algorithms for systematically enumerating mechanisms of various types with a given DOF and a given number of links. The enumeration technique produces millions of mechanisms even for small DOFs and link numbers. Sophisticated kinematic simulation methods are needed to find the behavior of each of these enumerated mechanisms.[3] Pure structural theories of synthesis result in a generate-and-test method for producing mechanisms, given input-output motion specifications. That is, mechanisms are enumerated and tested to check that they meet the given motion specification. This procedure is quite expensive, and it is generally difficult to exploit information about the desired motion to control the enumeration phase.

2. *Behavioral Theories*: These theories derive the structure of a mechanism from specifications of its output and input motions. Behavioral theories fall into two categories: compositional and non-compositional. *Compositional theories* of motion synthesis 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

[3] Rarely can we predict the motion characteristics of these mechanism graphs without instantiating their dimensions and running a complete kinematic simulation.

of large scale electro-mechanical systems. *Non-compositional* theories 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.

(a) *Compositional Theories*: Compositional theories solve the following problem: Given motion specifications (I, O) for the desired mechanism, a finite set $\{P_i\}, i > 0$ of primitive mechanisms with $O_i = P_i(I_i)$ where I_i and O_i are the input and output motions of primitive P_i, and a scheme for composing primitive mechanisms, find a composition $C = P_{i_1} \circ P_{i_2} \circ \cdots \circ P_{i_n}$ such that $O = C(I)$. Typically, the composition process requires efficient computation of I_i from P_i and O_i. This involves calculating the *regression* $I_i = P_i^{-1}(O_i)$. In the literature, we find specifications of P_i, I and O in specialized qualitative languages designed for specific synthesis problem classes. Expressiveness of the language and efficiency of algorithms for calculating P_i^{-1} are often at odds, and authors make different engineering tradeoffs along this spectrum.

Kota [19] was one of the first authors to propose a compositional theory of motion synthesis — in his scheme, motions are qualitatively described by vectors which capture pre-specified properties: e.g., a rotation has sense alone. He provides a calculus for deriving P_i^{-1} in his language. Since there isn't enough detail in Kota's qualitative language, the conceptual synthesis engine cannot be cleanly integrated with dimensional synthesis. Ulrich [29] also proposes a compositional technique for pre-parametric design. His design problems are specified qualitatively in the vocabulary of bond graphs [25] — effort and flow variables in electrical, mechanical and fluid media. The language of bond graphs is expressive enough for constraints on power flow but cannot express geometric constraints which are crucial in kinematic synthesis of mechanisms. The designs produced are bond graphs for single-input, single-output dynamical systems. The algorithmic focus is on optimizing these designs by function sharing. The constraint representations do not allow for easy specification of dimensional information – indeed, the system produces designs represented as abstract bond graphs. The refinement of these graphs to concrete mechanisms is not easily automatable. Hoover and Rinderle's [11] motion language is geared for describing pure rotations only. They can express geometric constraints, as well as power and torque constraints for rotary motions. Their calculation of P_i^{-1} is very efficient because their constraints are simple arithmetic equalities and inequalities.

Finger and Rinderle [5] use context-free and context-sensitive grammars for representing repeated structural elements in gear trains and bridge trusses and propose a synthesis scheme based on sentence generation in grammars. Joskowicz and Neville [14] design a complete language for describing complex motions built out of elementary motions and special predicates to capture effects like dwell, and reciprocation. This language is very expressive, however no algorithms for computing regressions of motions are provided. Their focus is on kinematic simulation – the generation of behavior descriptions from information about detailed structure of a mechanism.

The theory presented in this paper is a compositional theory of motion synthesis. It is similar in spirit to the theories proposed by Kota and Hoover and Rinderle. It allows for specifications of a richer set of constraints on motion than Kota and handles a wider range of motions than Hoover and Rinderle. We ground our composition process in the framework for configuration spaces. This allows us to formally characterize requirements on our qualitative motion language to guarantee that the designs calculated are correct. The efficiency of our goal-directed scheme for enumerating compositional designs stems from the fact that our constraint representation allows us to perform delayed instantiation of parameters. Each search path thus encodes a whole class of solutions. Pruning or accepting a path involves pruning or accepting a whole class of solutions. The procedure for handling constraints used in this paper is one of main sources of power for our synthesis method.

(b) *Non-Compositional Theories*: This category includes the work of Marshek and Kannapan [17], Prabhu and Taylor [26] and Joskowicz and Addanki [13]. Marshek and Kannapan design a qualitative motion language geared for solving synthesis problems involving rotational coupling between shafts. Design is performed by rewriting of motion descriptions. The space of motion description rewrites encodes the space of possible designs. The rewrite system has a large number of rules, which makes the automatic determination of the sequence in which to apply the rules, quite hard. A hand simulation of the rewrite system was shown to generate a new shaft coupling. The method is not scalable, but is useful as a tool for training human designers who have the intelligence to apply these rules appropriately. Prabhu and Taylor provide a scheme for synthesizing systems represented as bond graphs. Our use of configuration spaces to study mechanism design has been inspired by the work of Joskowicz and Addanki [13] on synthesizing part shapes. While [4, 15] have used configuration space analyses for kinematic simulation, our method and its symbolic compilation are completely automated methods for compositional kinematic synthesis.

3. *Functional Theories*: These theories work from intended function of a device and derive a structure that performs that function. An example is the design of a system to clean toner from film in the cleaning subsystem of a copier. Functional theories of synthesis posit intermediate behavioral specifications and then design structures that generate those behaviors. A wiping motion can be used to clean toner from a film. So can a shovel-like motion. The generation of conceptual possibilities for motion specifications that achieve a given function is very hard. Few functional theories exist in the literature: [7], [3] and [20] take kinematic function into account in the design of mechanism topologies.

1.3 Guide to Paper

This paper is organized as follows. In Section 2, 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 3. The composition operators form the basis for a rigorous specification and solution of the motion synthesis problem. The common types of kinematic synthesis problems are recast in our framework in Section 4. The general algebraic synthesis technique is presented in this section. In Section 5, we discuss tractable representations of the configuration space descriptions manipulated by our synthesis method. The algebraic composition computation can be abstracted as operations on symbolic predicates denoting qualitative motions. We then present constraint-based algorithms that employ qualitative motion descriptions. These algorithms have been implemented in CLP($\mathcal{R}$) [9] and we present examples of interesting syntheses in Section 6. Section 6.3 provides computational results on our implementation and explains why our program is very efficient in practice. In Section 7, we conclude by reiterating the main contributions of our paper and provide a discussion of future work on the specific problem of automating motion synthesis and the general problem of conceptual design.

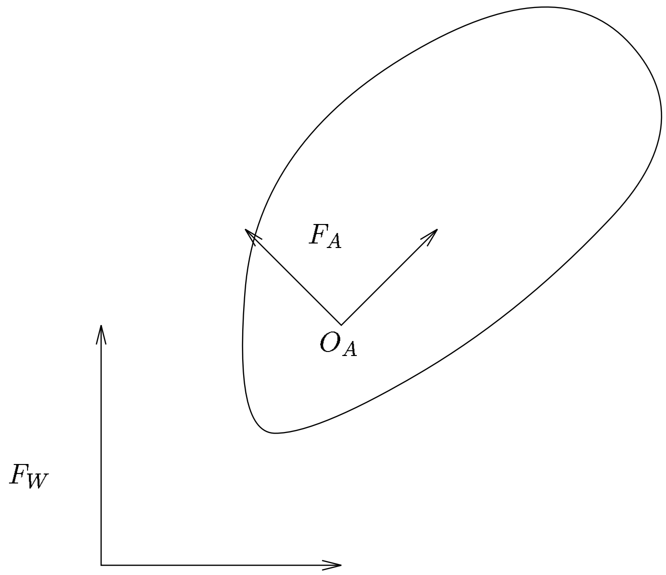


Figure 6: A Configuration of an Object

2 Configuration Spaces, Motions, and Mechanisms

We briefly review the concept of a configuration space before formally defining the motion of an object. The configuration of an object [21, pg. 8] is a specification of the position of every point on it relative to a fixed frame of reference. The objects we consider are compact subsets of the N-dimensional Euclidean space $\mathbf{R}^N$. 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 (figure 6).

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 $\mathcal{C}$ of all possible configurations of A.*

The configuration space $\mathcal{C}$ is intrinsically independent of the choice of F_A and F_W. However, the *representation* of $\mathcal{C}$ depends on these two choices. The configuration of a rigid body constrained to move on a plane can be represented by the triple (x, y, θ), where $(x, y) \in \mathbf{R}^2$ is the location of origin of F_A, and $\theta \in [0, 2\pi)$ is the angle between the x axes of F_A and F_W. The dimension of the configuration space in this case is three, because any representation of this configuration space requires the specification of three independent parameters. The dimension of the configuration space of an object is its number of degrees of freedom. A rigid body constrained to move on a plane has three degrees of freedom: two translational, and one rotational. Note that the configuration space is atemporal — that is, it is an unordered set of configurations of an object.

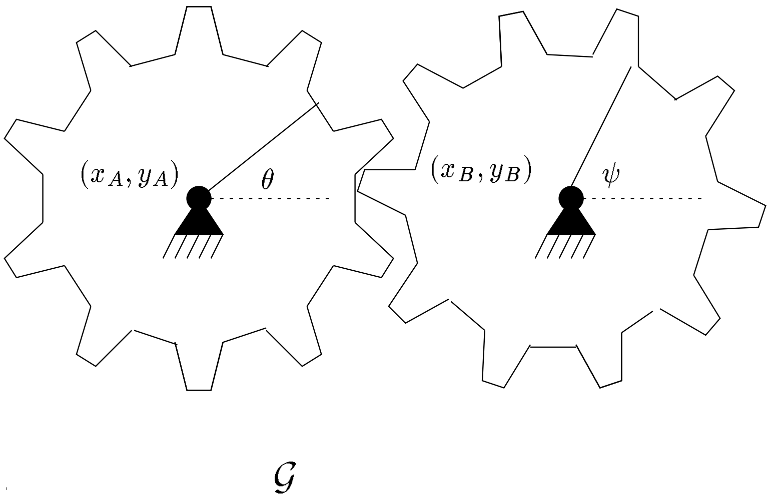


Figure 7: A meshed gear pair

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. For uniform rotation with angular velocity ω, $\theta(t) = \omega t$.

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 [27] 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.*[4]

[4] $\times$ stands for the cross product of two sets I and O. It is a set of pairs (i, o) where $i \in I$ and $o \in O$.

There is no temporal order on the configuration pairs defining an abstract mechanism. This is because the configurations of the input and the output links themselves are unordered sets, and an abstract mechanism is simply a subset of the cross products of these two sets.

Consider the meshed gear pair A and B on the xy plane with fixed centers at (x_A, y_A) and (x_B, y_B) in Figure 7. $\alpha > 0$ 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 $O = \{(x_B, y_B, \psi) \mid \psi \in [0, 2\pi)\}$,

$$\mathcal{G} = \{(x_A, y_A, \theta, x_B, y_B, \psi) \mid \psi = -\alpha\theta + \phi, (x_A, y_A, \theta) \in I, (x_B, y_B, \psi) \in O\} \tag{1}$$

This description[5] 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 $\mathcal{R}$.*

This definition is equivalent to the standard definition of the DOF of a mechanism as those motion variables that are independent after kinematic and structural constraints are satisfied.

Defining an abstract mechanism as a relation between two configuration spaces is precise, but sometimes unintuitive. This motivates us to define an abstract mechanism as a set of pairs of input and output *motions*.

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*

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

Note that $\mathcal{R}$ is a relation on configuration spaces and it relates the instantaneous positions of the input links with those of the output links. However, $\mathcal{R}'$ is a relation between two motions and it relates the motion of the input links with that of the output links. Recall that motions are

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

functions of time — they describe changes of configuration with time. Definition 5 describes all possible motion transformations of the mechanism. As an example consider a mechanism with an input link that rotates about the x axis and an output link that reciprocates along the y axis.

$$\mathcal{R} = \{(x_i, y_i, z_i, \theta, 0, 0), (x_o, y, z_o, 0, 0, 0) \mid y = g(\theta)\}$$

The configuration space relation that defines this mechanism relates the angular orientation of the input link to the position of the output link.

$$\mathcal{R}^{'} = \{(M_i, M_o) \mid M_i(t) = (x_i, y_i, z_i, \theta(t), 0, 0), M_o(t) = (x_o, y(t), z_o, 0, 0, 0), y(t) = g(\theta(t))\}$$

By defining this as a relation on motions, we can reformulate the description to be explicit functions of time. If $\theta(t) = wt$; that is, the input crank rotates uniformly about the x axis with angular velocity ω, then the position of the output link is $g(\omega t)$, that is the output link reciprocates along the y axis at a frequency computable from $g(\omega t)$. The two definitions are related since we have shown how to calculate M_o from $\mathcal{R}$ and M_i. We shall use both definitions in the later sections.

The implementation of an abstract mechanism—a concrete mechanism, is a geometric description of links, and joints between links that implements a relation between configuration spaces. 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 labelled according to joint type [27]. 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.

3 Composition of Mechanisms

Complex mechanisms are composed out of simpler ones. Composition constructs the relation (I_1, O_2) from (I_1, O_1) and (I_2, O_2) by imposing the equality constraint $O_1 = I_2$. In other words, we require that the configuration space of the output link of one mechanism be the same as the configuration space of the input link of the other. 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.[6]

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*

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

The 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 $\mathcal{G}_2$.

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

The gear ratios α_1 and α_2 are both positive. From $\mathcal{GT}$, we can derive the fact that $\psi' = -\alpha_2(-\alpha_1\theta + \phi) + \phi'$.

Using the alternate definition of abstract mechanism, Definition 5, we can express the composition of $\mathcal{R}_3 = \{(M_i, M_o)\}$ and $\mathcal{R}_4 = \{(M_i', M_o')\}$ as:

$$\mathcal{R}_3 \bowtie \mathcal{R}_4 = \{(M_i, M_o') \mid (M_i, M_o) \in \mathcal{R}_3 \wedge (M_i', M_o') \in \mathcal{R}_4 \wedge M_o = M_i'\}$$

[6] This composition operation is to be distinguished from that introduced in [12]. Joskowicz treats composition between two configuration spaces and not between *relations* on configuration spaces. His composition scheme is suited for kinematic simulation.

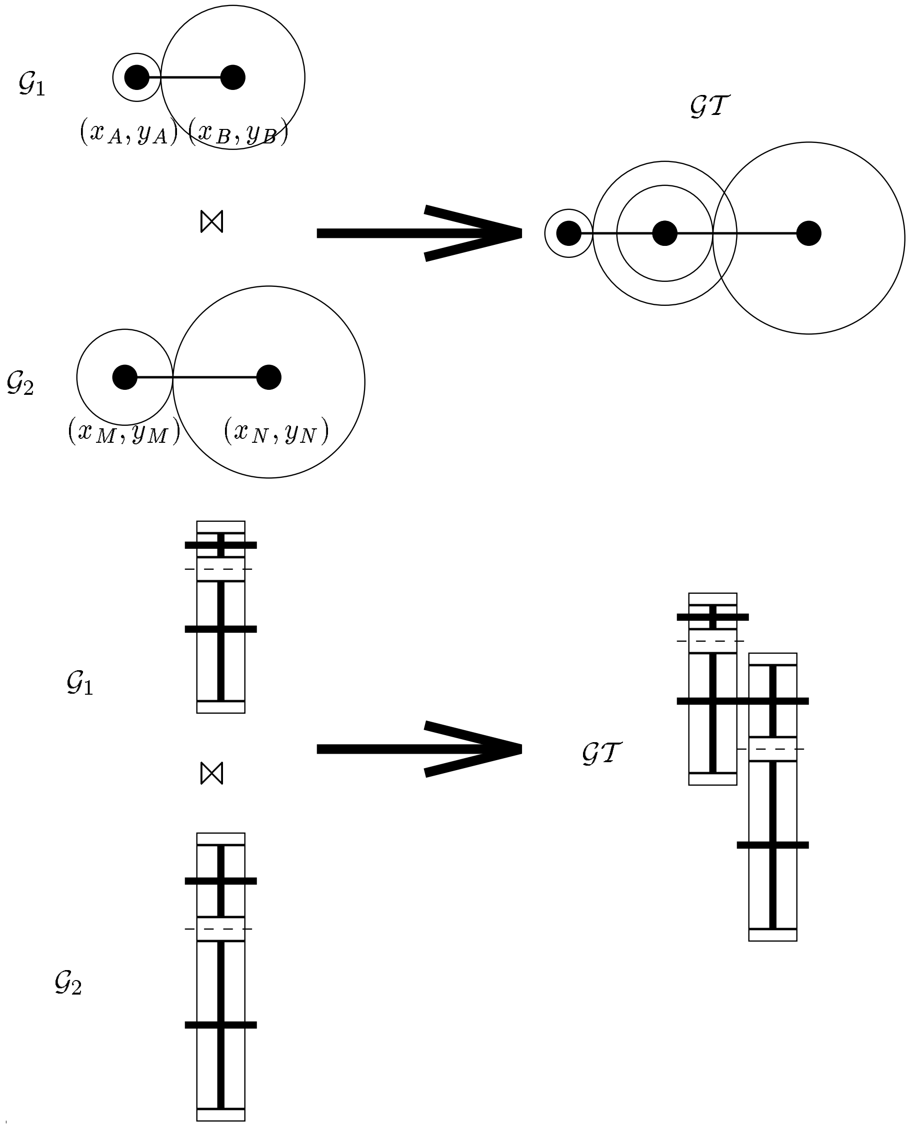


Figure 8: Composing two gear pairs

This description of composition requires the output motion of one mechanism to be the same as 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 $\mathcal{R}_2$ respectively 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. For example, in figure 8, we show the composition of the implementations of $\mathcal{G}_1$ and $\mathcal{G}_2$ to yield an implementation of $\mathcal{GT}$.

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 denoting rigid connections between output links in V_1 and corresponding input links in V_2.*

We now derive a more complex composition of two multi-DOF abstract mechanisms and their concrete counterparts shown in Figure 9, to demonstrate the generality of our algebraic scheme. This example demonstrates that the abstract mechanisms that can be represented and composed can have any number of degrees of freedom.

We start with two cylinders forming cylindrical joints (1 rotational and 1 translational degree of freedom each) with the ground link, and we compose them to form a 1-DOF mechanism with one translational degree of freedom. The configuration space relation implemented by each cylinder is of the form $\{(x, y, z, \phi, \theta, \psi, x, y, z, \phi, \theta, \psi)\}$, i.e., the inputs and outputs are the same. So for clarity, we simplify the description as $\{(x, y, z, \phi, \theta, \psi)\}$. The concrete implementation of this composition involves connecting the reference points[7] shown in Figure 9 on the two cylinders. The local coordinate frame on the left cylinder is at $(0, y, 0)$ and it is rotated about the y axis by θ. The local coordinate frame on the right cylinder is at $(0, y', 0)$ and it is rotated about the y axis by θ'. We describe the position of the center of rotation of this cylinder with respect to the local frame of reference in $\mathcal{C}_2$. When $r \neq 0$, where r is the radius of the cylinder on the right, the composition is:

$$
\begin{aligned}
\mathcal{C}_1 &= \{(0, y, 0, 0, \theta, 0) \mid y \in \mathbf{R} \wedge \theta \in [0, 2\pi)\} \\
\mathcal{C}_2 &= \{(r(1 + \cos\theta'), y', r\sin\theta', 0, \theta', 0) \mid y' \in \mathbf{R} \wedge \theta' \in [0, 2\pi)\} \\
\mathcal{C}_1 \bowtie \mathcal{C}_2 &= \{(0, y, 0, 0, \theta, 0) \mid y \in \mathbf{R} \wedge r(1 + \cos\theta') = 0,
\end{aligned}
$$

[7] The representations of C-spaces depends on the choice of reference points and local coordinate frames, which is typically chosen so as to simplify the algebraic description of the C-space itself.

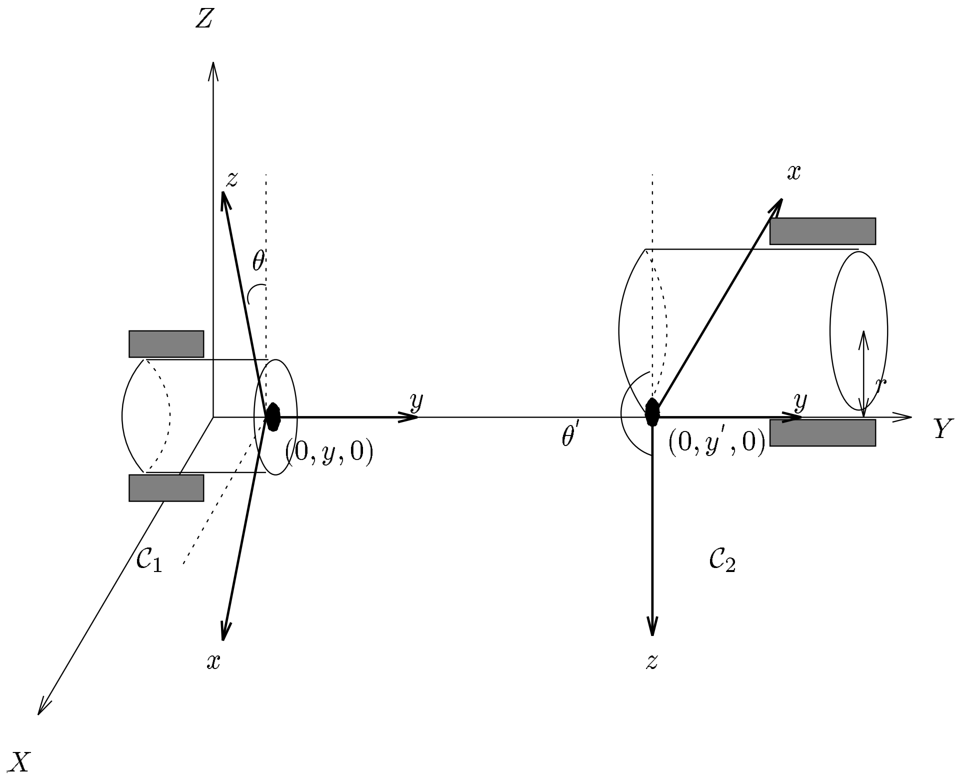


Figure 9: Composing two 2-DOF mechanisms

$$r \sin\theta' = 0, y = y', \theta = \theta')\}$$

We can derive $\theta = \pi$ from the constraints on $\mathcal{C}_1 \bowtie \mathcal{C}_2$.

4 The Synthesis Problem

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, we are given constraints on the motions 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. The standard classification of kinematic synthesis problems [22] can be recast in our framework as follows. Readers unfamiliar with this classification can omit the itemization below and proceed to the description of our solution method.

- **Path Generation**: Given P, a 1-dimensional subset of $\mathbf{R}^2$, find a concrete mechanism realizing $\mathcal{R} \subseteq I \times O$ such that $\Pi_{xy}(O) = P$. (Π_{xy} is the *projection* of a configuration onto the xy plane. $\Pi_{xy}(x, y, \theta) = (x, y)$. It is extended naturally to project sets of configurations.)

- **Rigid Body Guidance**: Given a sequence S of configurations, find a concrete 1-DOF mechanism realizing $\mathcal{R} \subseteq I \times O$ such that every element in S is in O, and their order is preserved in O.

- **Function Generation**: Given a function $f : I \rightarrow O$, find a concrete mechanism realizing a functional relation $\mathcal{R} \subseteq I' \times O'$ such that $I \subseteq I'$, $O \subseteq O'$, and for all $i \in I$, $(i, f(i)) \in \mathcal{R}$.

Path generation and rigid body guidance problems specify constraints on O alone, leaving the designer to pick I and thus $\mathcal{R}$. Function generation problems impose more constraints on $\mathcal{R}$; in particular they identify a subset of $I \times O$ such that $O = f(I)$.

4.1 Algebraic Synthesis Method

We view mechanisms as motion transformers [18]. 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 compositions of primitive relations $\mathcal{R}_p$ until the composition satisfies the specification. The predicate **SatisfyingDesign?**$(\mathcal{M})$ returns true if and only if $\mathcal{M}$ is a composition of primitives that meets the constraints on $\mathcal{R}$. **Synthesize** returns a sequence of primitive relations. We start by invoking **Synthesize**($\mathcal{I}$).

Synthesize($\mathcal{M}$) = if **SatisfyingDesign?**($\mathcal{M}$) then return [ ]
else select $\mathcal{R}_p$, a primitive abstract mechanism such that $\mathcal{R}_p \bowtie \mathcal{M}$ is non-empty
return [$\mathcal{R}_p$,**Synthesize**($\mathcal{R}_p \bowtie \mathcal{M}$)]

Concrete designs are produced by finding implementations for the compositions of primitive relations found by the procedure **Synthesize**.

$$\mathbf{Refine}(\mathcal{R}) \quad = \quad \begin{cases} \text{implementation}(\mathcal{R}) & \text{if } \mathcal{R} \text{ is primitive} \\ \mathbf{Refine}(\mathcal{R}_1) \bullet \mathbf{Refine}(\mathcal{R}_2) & \text{if } \mathcal{R} = \mathcal{R}_1 \bowtie \mathcal{R}_2 \end{cases}$$

For each primitive motion relation we non-deterministically[8] pick an implementation. We use Definition 8 to compose two 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:* **Refine(Synthesize($\mathcal{I}$))** *with a* **SatisfyingDesign?** *predicate is sound. That is, if it produces a design, it will satisfy the specified motion constraints.*

The theorem follows from the correctness of the **SatisfyingDesign?** predicate that checks that the composition of primitive relations satisfies the specification of $\mathcal{R}$. By the construction in Definition 8, we can prove the correctness of **Refine**.

To prove completeness[9], 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, we have abstract mechanisms that can convert uniform rotations and translations into uniform as well as non-uniform rotations and translations. The set of concrete implementations we have include gear

[8] This means we pick one out of a number of choices randomly.
[9] A set of primitive motion relations is complete if it can synthesize any algebraic relation on configuration spaces.

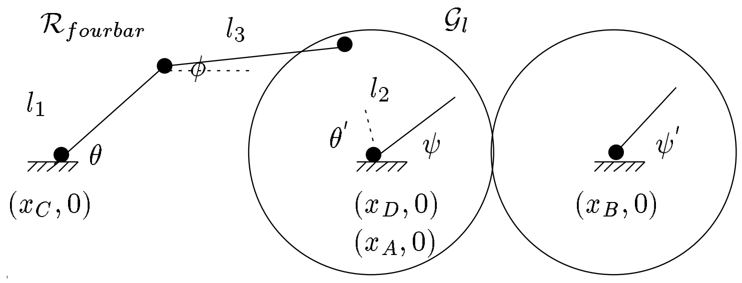


Figure 10: Composing $\mathcal{R}_{fourbar}$ and $\mathcal{G}_l$

pairs, racks and pinions, simple linkages, worm-spur pairs, skotch yokes, etc. Additional abstract mechanisms corresponding to motion relations with concrete implementations can be added very easily to our design system.

We now discuss the computational complexity of the **Synthesize** and **Refine** procedures. 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.

To see this consider $R_{fourbar}$, a specification of the motion relation implemented by a class of planar four bar linkages: conversion of a uniform rotation around the z axis at $(x_C, 0)$ into a nonuniform rotation around the z axis at $(x_D, 0)$. We will compose it with a linear rotation transformer $\mathcal{G}_l$ as shown in Figure 10.

$$\begin{aligned}
R_{fourbar} &= \{(x_C, 0, \theta, x_D, 0, \theta') \mid x_D - x_C = l_1 \cos\theta + l_2 \cos\theta' + l_3 \cos\phi, \\
&\quad 0 = l_1 \sin\theta - l_2 \sin\theta' + l_3 \sin\phi\} \\
\mathcal{G}_l &= \{(x_A, 0, \psi, x_B, 0, \psi') \mid \psi' = -\alpha\psi + \beta,\ \alpha > 0\} \\
R_{fourbar} \bowtie \mathcal{G}_l &= \{(x_C, 0, \theta, x_B, 0, \psi') \mid \theta' = \psi,\ x_D = x_A,\ x_D - x_C = l_1 \cos\theta + l_2 \cos\theta' + l_3 \cos\phi, \\
&\quad 0 = l_1 \sin\theta - l_2 \sin\theta' + l_3 \sin\phi,\ \psi' = -\alpha\psi + \beta,\ \alpha > 0\}
\end{aligned}$$

To verify that this composition satisfies $\mathcal{R}$, we will need to solve these equations to eliminate intermediate variables like ϕ. The verification step has the same complexity as that of computing each composition.

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.

5 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 which is useful in the formulation of motion synthesis problems when the entire description of the motion is not available. Second, they allow for potential efficiency gains in performing compositions 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 [13, 16, 19, 29] in its use of predicate calculus. However, unlike these approaches, but in common with [12], 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 $\bowtie$-*preservation* property that a qualitative motion language must satisfy in order to generate correct syntheses.

A qualitative motion language can be characterized by a homomorphic mapping A[10] which picks out specific properties of a motion i.e., there is a mapping A from a motion relation to its qualitative description. 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

[10] Let S be a set with a binary operation * defined on it and let T be an other set with a binary operation + defined on it. A function $f : S \rightarrow T$ is a homomorphism if and only if $f(a * b) = f(a) + f(b)$.

$$
\begin{tikzcd}[column sep=huge, row sep=huge]
\mathcal{R}_1, \mathcal{R}_2 \arrow[r, "\bowtie"] \arrow[d, "A"'] & \mathcal{R}_3 \arrow[d, "A"'] \\
A(\mathcal{R}_1), A(\mathcal{R}_2) \arrow[r, "\hat{\bowtie}"] & A(\mathcal{R}_3)
\end{tikzcd}
$$

Figure 11: $\bowtie$-preservation by abstraction A

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 $A(\mathcal{R}_1)$ and $A(\mathcal{R}_2)$ as in Figure 11, for motion relations $\mathcal{R}_1$ and $\mathcal{R}_2$, we have an adequate language. In other words, we need a language for computing the description of $\mathcal{R}_1 \bowtie \mathcal{R}_2$ from the descriptions of $\mathcal{R}_1$ and $\mathcal{R}_2$. The formal property is called $\bowtie$-preservation and requires the specification of $\hat{\bowtie}$—the composition operation in the abstract language defined by A.

Definition 12 *The mapping A is $\bowtie$-preserving if*

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

Note that this constraint places restrictions on the definition of $\hat{\bowtie}$ — the implementation of $\bowtie$ for abstraction A. We illustrate the $\hat{\bowtie}$ computation using the gear train example of Section 3. $\mathcal{G}_1$ and $\mathcal{G}_2$ transform rotary motion to another rotary motion with a different speed and sense. The predicates below reformulate the algebraic descriptions of rotation provided earlier.

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

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

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

To implement $\bowtie$ by $\bowtie$, 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. $T_{trans}(x_M, y_M) = (x_B, y_B)$
2. $T_{trans}(x_N, y_N) = (x_F, y_F)$

We calculate the rigid transformation T_{trans} (in this case, a pure translation) which moves the rotation about the z axis from (x_M, y_M) to (x_B, y_B). We then apply that transformation to the output of $\mathcal{G}_2$. In this example, no computational advantage is obtained over the algebraic descriptions by the use of our qualitative reformulation. This is because, *for the purposes of the composition computation*, the information captured by the qualitative vocabulary is contained exactly in the equations manipulated earlier.

For the simple language of rotations and rectilinear translations introduced above, $\bowtie$ "unifies" motions M_i and M_j by calculating the generalized rigid transformation T such that $T(M_i) = M_j$. 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). This unification captures the composition calculation in C-space and the abstract description filters exactly the properties of interest.

We will call a mechanism *linear* if its output motion is a linear function of its input motion. For example, a spur-gear pair and a worm-spur pair are implementations of linear rotary motion converters. The frequency of motion is preserved by a linear mechanism.

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 shown in Figure 12. The abstract motion relation implemented by the slider crank is:

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

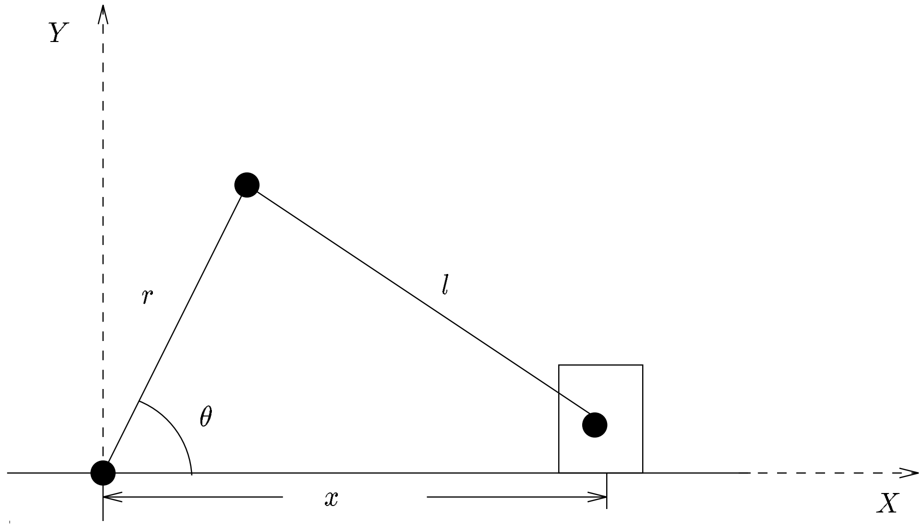


Figure 12: A Slider Crank Mechanism

$$\mathcal{S}_q \quad = \quad \begin{array}{ll} \textbf{Input}\text{: rotation} & \textbf{Output}\text{: translation} \\ \text{center: } (0,0) & \text{range: } 2r \\ \text{angular\_velocity: } S \text{ rev/sec} & \text{frequency: } S \end{array} \tag{3}$$

Our qualitative description only picks out the motion interval and frequency of the slider, because these are the only characteristics of the slider's motion that we can describe in our vocabulary. 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 $\bowtie$ calculation. We no longer need to solve the non-linear equation 2 to test **SatisfyingDesign?**.

In general, if the configuration space relations $\mathcal{R}_1$ and $\mathcal{R}_2$ are linear (representable by linear constraints), the $\bowtie$ defined above, permits A to be $\bowtie$-preserving. To better understand the $\bowtie$-preservation property, it is useful to consider the example in Figure 13 where $\bowtie$-preservation is violated.

The output of the crank rocker in Figure 13 is a rectilinear translation (reciprocation) which is the input to the skotch-yoke mechanism. Our $\bowtie$ construction unifies the two motions as long as their ranges, speed, and axes coincide. Unfortunately, an analysis of the underlying C-space relations reveals that there is at most one possible position where the two motions intersect, thus

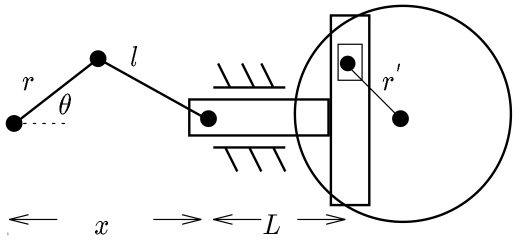


Figure 13: An example that shows join preservation violation

the mechanism jams (becomes rigid). Both the crank-rocker and skotch-yoke mechanisms are non-linear.

The specific $\bowtie$ that we developed above is inadequate for handling non-linear mechanisms in a general way. To guarantee $\bowtie$-preservation for the specific $\bowtie$ and A combination introduced here, we ensure that non-uniform motions are not composed. This can be done with the restriction that nonlinear mechanisms only take uniform motions as input. Since no non-uniform motion can be transformed to a uniform motion within the primitive abstract mechanisms that our system possesses, composite mechanisms can contain at most one nonlinear mechanism. Most mechanisms used in practice satisfy these restrictions. An important exception occurs when a nonuniform mechanism is composed with a copy of itself, then the resulting mechanism may be linear. For example, automotive transmissions have compositions of two Hooke joints with the same angles.

Unlike other efforts [15, 16, 19] to design qualitative motion vocabulary in the context of synthesis, our qualitative descriptions are derived as abstractions of underlying relations on configuration spaces. This allows us to understand what the qualitative descriptions mean, what loss of expressive power is entailed by their use, and whether we obtain computational efficiency from them. In particular, the $\bowtie$-preservation property permits the assessment of the correctness of a given qualitative motion vocabulary with respect to a class of synthesis tasks.

In the next section, we show how qualitative motion languages can be used to generate efficient constraint-solving schemes for kinematic synthesis.

6 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 $\mathcal{A}_1, \ldots, \mathcal{A}_n$ 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 $m_i \in i$ means that the motion m_i is in the class of motions described by the qualitative specification i.

We adapt the **Synthesize** procedure developed in Section 4 to handle constraints on qualitative motion descriptions. To make it efficient, we transform the naive generate-and-test scheme to a goal-directed procedure 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 which compute input motions of primitive abstract mechanisms given output motions. We will illustrate the algorithms in the context of the simple motion description system introduced in Section 5.

6.1 Synthesizing single input, single output mechanisms

The recursive algorithm for synthesizing single input, single output mechanisms is shown in Table 1. We implement the synthesis method for our language as a depth-bounded, goal-directed, depth-first backward chainer in CLP($\mathcal{R}$). The operational model of CLP($\mathcal{R}$) 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.

In our program, we differentiate between linear and nonlinear primitive abstract mechanisms. 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:

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 $T(\mathcal{M})$ 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 $T(\mathcal{M})$.

3. Return [$T(\mathcal{M})$, **SISO_Synthesize**(i,q)].

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

```
mechanism(gear_pair(3,5),                    % name
          % scaling factor. For this mechanism, it relates the angular velocity
          % of the gears
          linear(-3/5),
          % input motion type, the two triples are the center
          % and direction of axis of rotation respectively.
          [rotation((0,0,0),(0,0,1))],
          [rotation(((3+5)/2,0,0),(0,0,1))]).% output motion type
```

For a nonlinear primitive mechanism, we store its name, its input motion, which must be uniform (rotation or translation with constant velocity), and its output motion. For instance, the abstract nonlinear mechanism corresponding to a crank rocker is represented as:

```
mechanism(crank_rocker(L),                                        % name
          nonlinear,
          [rotation((0,0,0),(0,0,1)), angular_velocity(F)],       % uniform input motion
          [rotation((L,0,0),(0,0,1)), frequency(F), range(R)])    % output motion
          :- R > 0, R < π.                                        % constraints
```

The top-level invocation of the synthesis function is: *synthesize(input motion, output motion, Design, depth bound)*. Note that *Design* is a variable that gets instantiated when *synthesize* terminates. The base case of the synthesis occurs when the input motion i is equal to the output motion o: this is established by solving arithmetic constraints that are generated when the motions are unified.

```
%%% Base case
```

```
synthesize(Input_motion, Output_motion, Design, Depth) :-
        motion_equal(Input_motion, Output_motion).
```

`motion_equal` is a predicate testing whether two motion descriptions are equivalent. Its actual implementation depends on the specific motion language. For our language, one of the rules for checking motion equivalence is

```
motion_equal([rectilinear_translation(X,Y,Z), speed(S)], % (X,Y,Z) is the direction of translation
           [rectilinear_translation(-X,-Y,-Z), speed(-S)]).
```

This rule says that two rectilinear translations are the same if they have opposite directions and opposite speeds. For each type of motion, we have corresponding rules for deciding equivalence.

The recursive step of synthesis first involves non-deterministic 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 $T(o_p) = o$. We need to find $T(i_p)$, the input motion of the primitive after rigid transformation. $T(i_p)$ can be calculated very easily from the type of input motion and the scaling factor that relates the input and output motions of the mechanism. Then, the new synthesis problem $i, T(i_p)$ needs to be solved.

```
%%% selecting a linear primitive mechanism
synthesize(Input_motion, Output_motion, [(Name,Rigid_transformation)|Design], Depth) :-
        Depth > 0,                       % check depth bound
        %% select a linear mechanism
        mechanism(Name, linear(Factor), Primitive_input_type, Primitive_output_type),
        %% find the rigid transformation
        transformation(Primitive_output_type, Output_motion, Rigid_transformation),
        %% find T(i_p)
        linear_apply(Rigid_transformation, Factor, Primitive_input_type, NewGoal),
        %% recursive call
        synthesize(Input_motion, NewGoal, Design, Depth-1).
```

If, however, the primitive mechanism chosen is not linear, the complete input and output motion description, i_p and o_p, will be given. We solve for the rigid transformation T directly. $T(i_p)$ is computed simply by applying T to the given input motion.

```
%%% selecting a nonlinear mechanism
synthesize(Input_motion, Output_motion, [(Name,Rigid_transformation)|Design], Depth) :-
        Depth > 0,
```

```
mechanism(Name, nonlinear, Primitive_input_motion, Primitive_output_motion),
transformation(Primitive_output_motion, Output_motion, Rigid_transformation),
nonlinear_apply(Rigid_transformation, Primitive_input_motion, NewGoal),
synthesize(Input_motion, NewGoal, Design, Depth-1).
```

Consider the design of the windshield wiper introduced in Section 1. The problem was specified as follows:

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

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

Input motion i_p	**Output motion** o_p
rotation	rotation
center: (0,0,0)	center: (L,0,0)
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 aligns 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 3×3 rotational matrix. Let ψ, ϕ, θ be rotational angles about the z, x, and y axis respectively. We have

$$R = \begin{pmatrix} \cos\psi\cos\theta - \sin\phi\sin\psi\sin\theta & -\cos\theta\sin\psi - \cos\psi\sin\phi\sin\theta & \cos\phi\sin\theta \\ \cos\phi\sin\psi & \cos\phi\cos\psi & \sin\phi \\ -\cos\theta\sin\phi\sin\psi - \cos\psi\sin\theta & -\cos\psi\cos\theta\sin\phi + \sin\psi\sin\theta & \cos\phi\cos\theta \end{pmatrix}$$

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.

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

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

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

The rigid transformation consists of 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 find a primitive mechanism to transform the specified input motion i to the intermediate motion generated. 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 the worm-spur pair can be matched with the input description i yielding $F_1 = 20$. This generates the derived constraint $F = 20/\alpha$. Note that dimensional constraints have been generated during the synthesis. 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 in Figure 14 accumulates simple algebraic constraints which are solved incrementally during the synthesis. The constraint programming language CLP($\mathcal{R}$) [9] is used to implement the algorithm. CLP($\mathcal{R}$) has mechanisms for solving and managing algebraic constraints. Graphical outputs are produced via an interface to Mathematica[11].

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

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. However, in practice, the algorithm is very efficient. This issue is discussed in Section 6.3. Some other designs produced by our system are in Figure 19 and Figure 15. A physical prototype of the clock design in Figure 15 is in Figure 16.

6.2 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

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

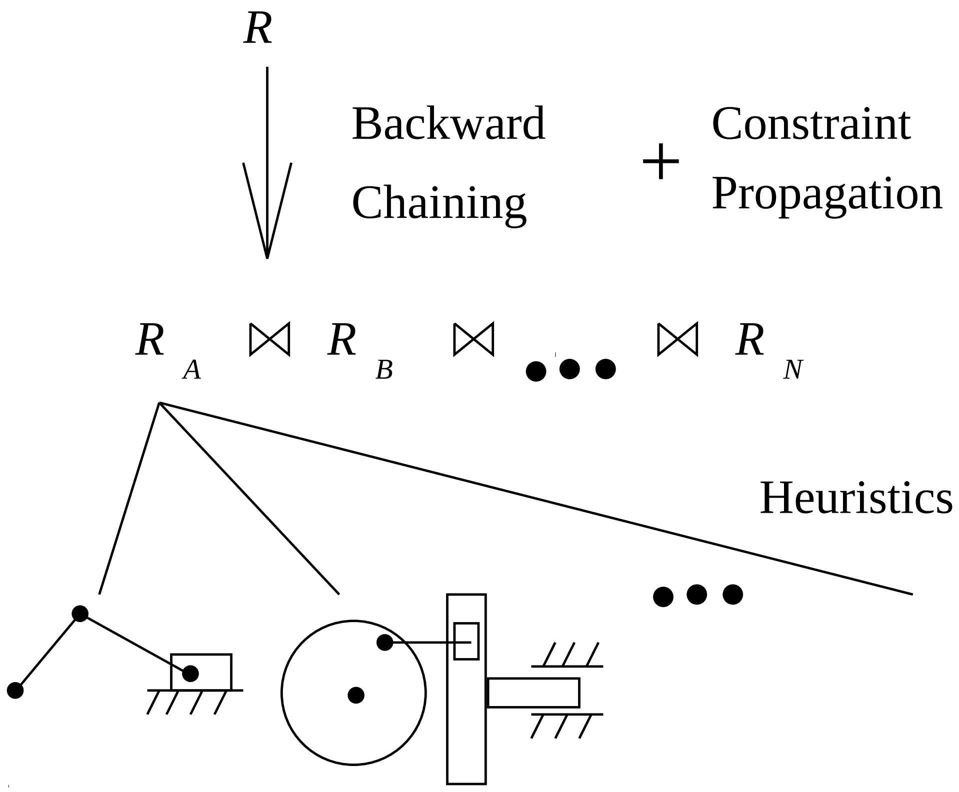


Figure 14: Our approach to motion synthesis

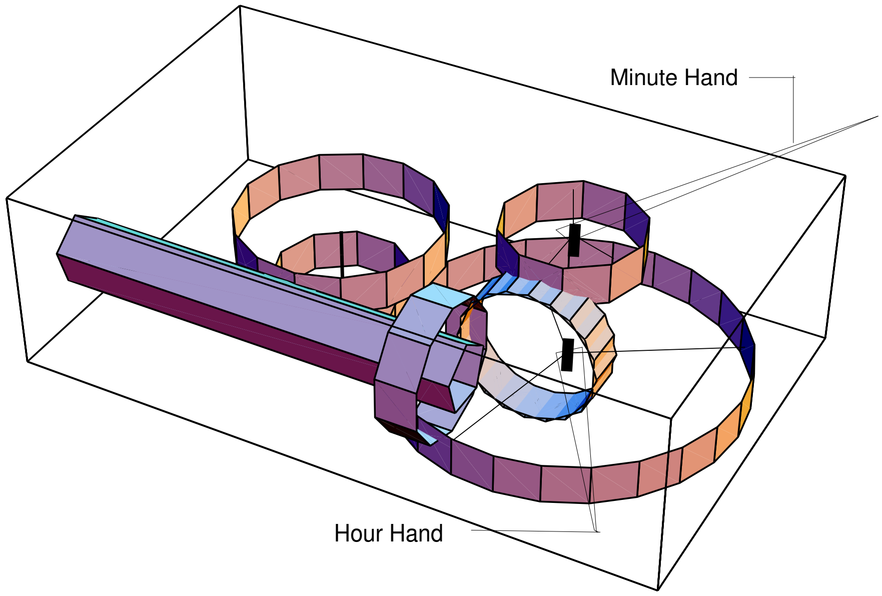


Figure 15: Clock design

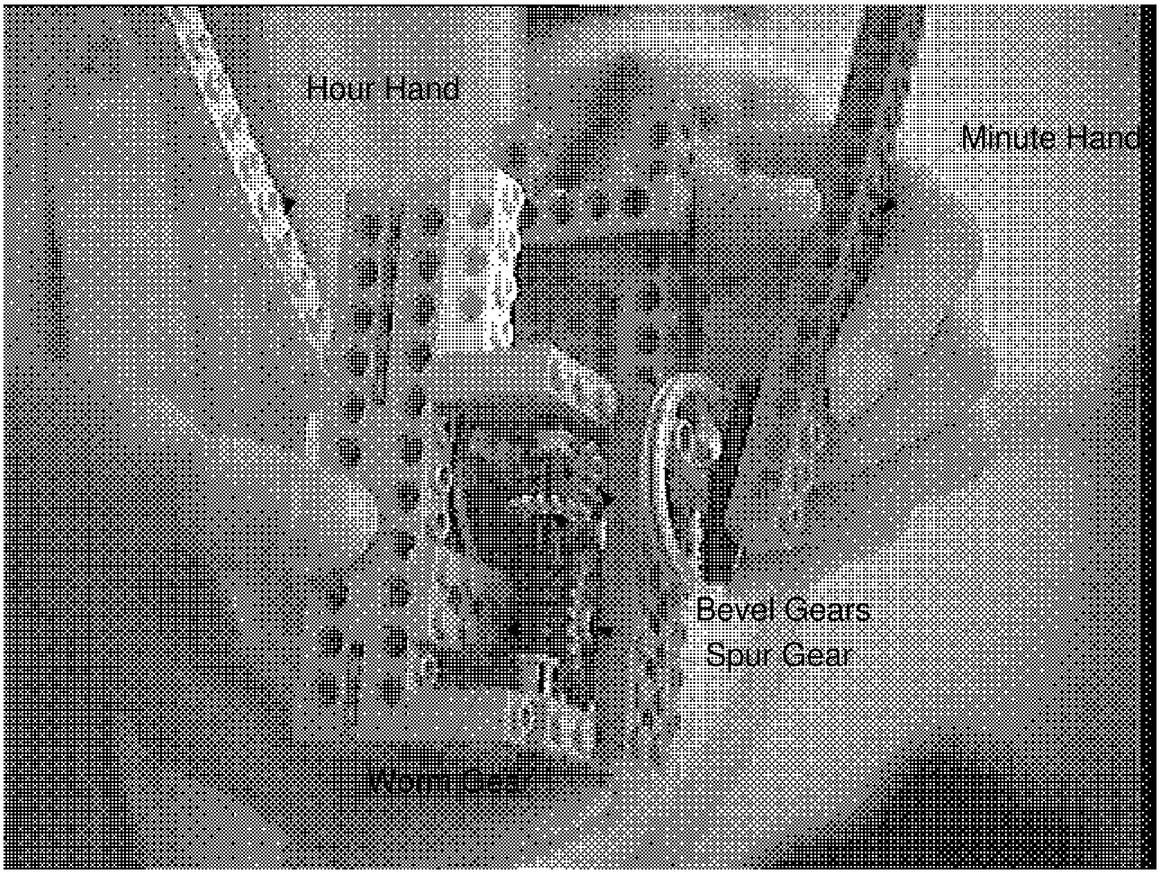


Figure 16: A Technics Lego implementation of the clock in Figure 15

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, $\mathcal{A}_1^1, \ldots, \mathcal{A}_{k_1}^1, \mathcal{A}_1^2, \ldots, \mathcal{A}_{k_n}^n$, which when composed satisfies the input-output specification, i.e.,

$$\forall 1 \leq j \leq n. \forall m_i \in i. \exists m^j \in o^j.(m_i, m_o^j) \in \mathcal{A}_1^j \bowtie \ldots \bowtie \mathcal{A}_{k_j}^j$$

We require that for all j, $\mathcal{A}_1^j \bowtie \ldots \bowtie \mathcal{A}_{k_j}^j$ will transform motions described by i into motion described by o^j.

SIMO_Synthesize$(i, o^1, \ldots, o^n)$

for j from 1 to n do
  $\mathcal{A}_1^j, \ldots, \mathcal{A}_{k_j}^j \leftarrow$ **SISO_Synthesize**(i, o^j)
/* optimization */
for j from 1 to n do
  for each $\mathcal{A}_k^j$ in $\mathcal{A}_1^j, \ldots, \mathcal{M}_{k_j}^j$ do
    for l from j+1 to n do
      for each $\mathcal{A}_m^l$ in $\mathcal{M}_1^l, \ldots, \mathcal{A}_{k_l}^l$ do
        if **InputMotion**$(\mathcal{A}_m^l) =$ **InputMotion**$(\mathcal{A}_k^j)$ then
          delete $(\mathcal{A}_1^l, \ldots, \mathcal{M}_{m-1}^l)$
          merge the inputs of $\mathcal{A}_m^l$ and $\mathcal{M}_k^j$¡

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

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 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 **InputMotion**$(\mathcal{A})$. In Figure 17 we show how the common reciprocation motion can be shared across two branches of a single-output, two-output design problem. When we merge the inputs of $\mathcal{A}_m^l$ and $\mathcal{A}_k^j$ in the last step of **SIMO_Synthesize**, the input links of $\mathcal{A}_m^l$ and $\mathcal{A}_k^j$, as well as the output links of $\mathcal{A}_{k-1}^j$ are

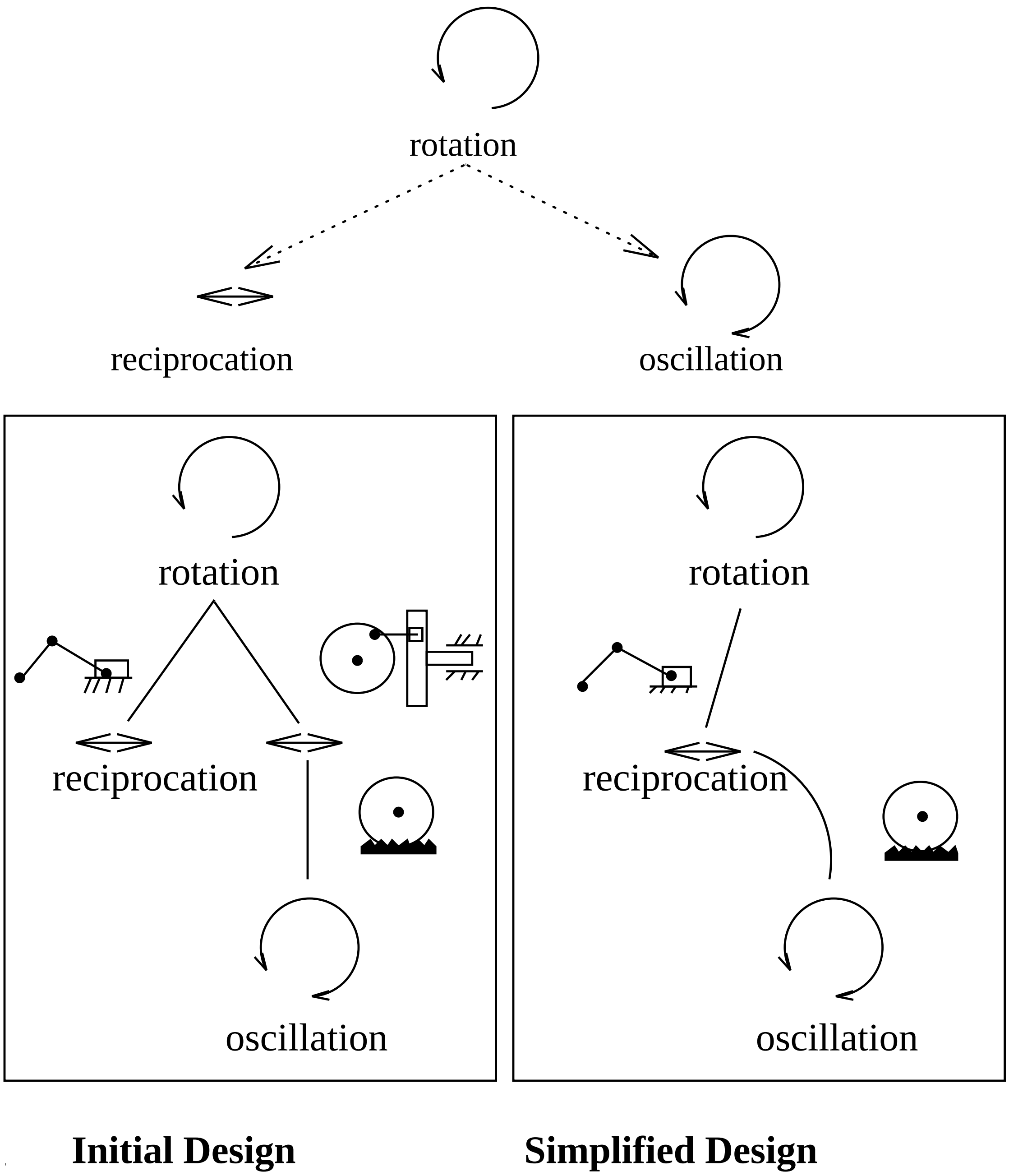
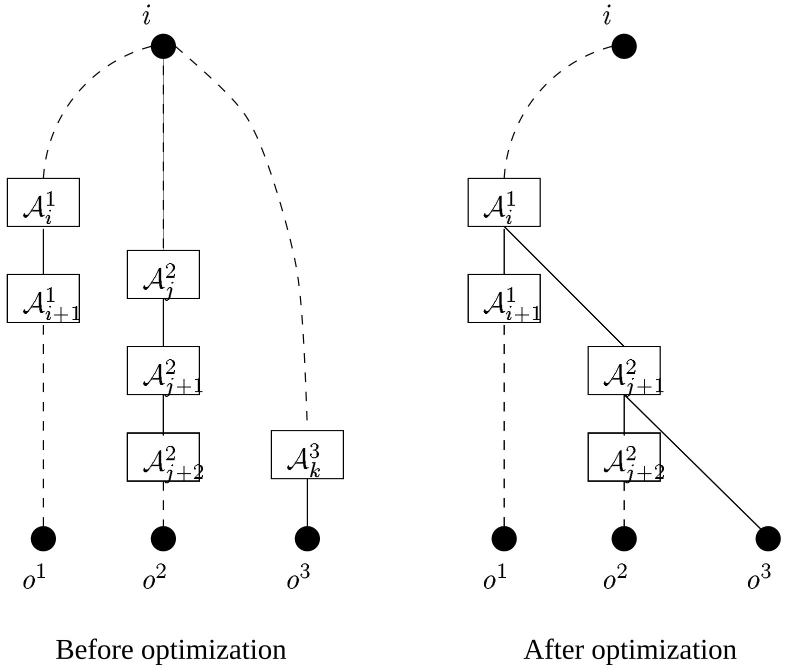


Figure 17: An example optimization of a SIMO mechanism



$\mathbf{InputMotion}(\mathcal{A}_{i+1}^1) = \mathbf{InputMotion}(\mathcal{A}_{j+1}^2)$
$\mathbf{InputMotion}(\mathcal{A}_{j+2}^2) = o^3$

Figure 18: Optimization of single input, multiple output mechanisms

rigidly connected together — an example is shown in Figure 18 . 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. The complexity of the optimization stage is p^2 where p is the number of primitives in the unoptimized design.[12]

Theorem 3 *Algorithm (***SIMO_Synthesize***$(Q_i, (Q_o^1, \ldots, Q_o^n)$) is sound: i.e, it designs concrete mechanisms that satisfies the motion specifications $(Q_i, (Q_o^1, \ldots, Q_o^n))$.*

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

[12]This is because we basically match the input motion of every primitive in the design against that of other primitives.

our current implementation, the class of mechanisms synthesizable are fixed-topology, single-degree of freedom mechanisms with at most one non-linear 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 a design is needed to ensure that the specific A developed here is $\bowtie$-preserving. 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 [13] for synthesizing novel shapes into our design system to automatically extend our library of primitive abstract mechanisms.

6.3 Computational Experiments

We tested the performance of both the SISO and the SIMO algorithms. The worst case branching factor for both searches is around 10, corresponding to the number of primitive motion relations. In practice, the average branching factor is much smaller (around 4 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 representations 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 under a second on a Sparcstation 1+. In Figure **??**, we measure

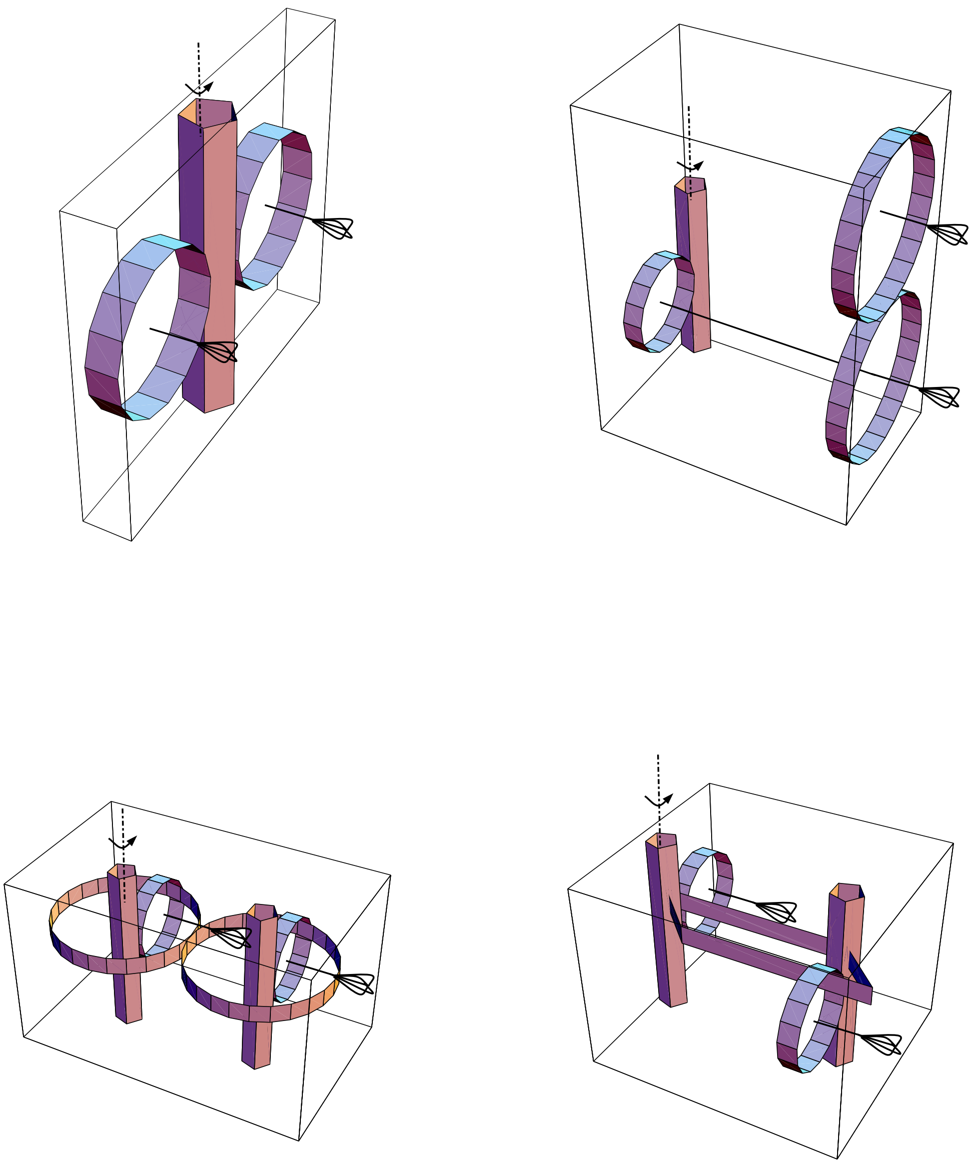


Figure 19: Eggbeater designs

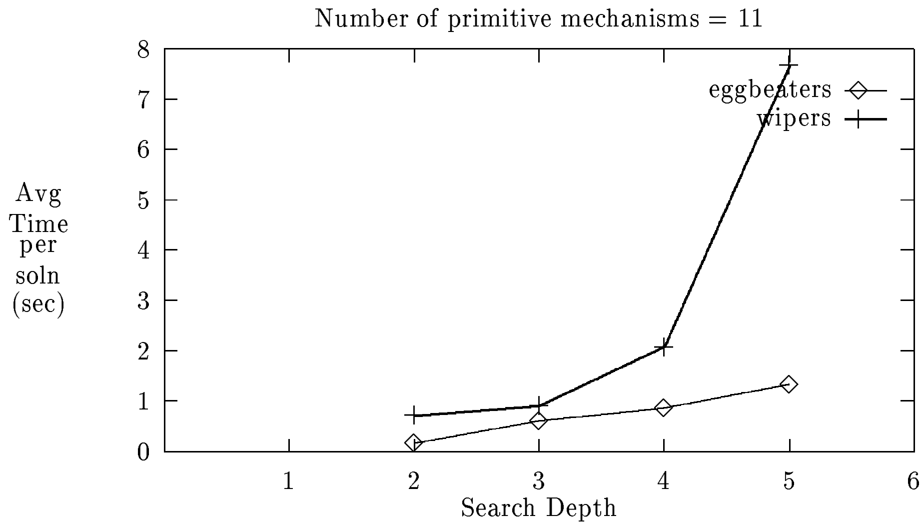


Figure 20: Performance Graph I

the number of solutions at a given search depth for a given configuration of primitive mechanisms for two design problems discussed in this paper: design of wipers and eggbeaters. Note that the number of solutions to each design rises exponentially with depth for these two designs. It is not clear that all design problems have this characteristic. The quality of solutions at any depth are not all alike, if we use criteria other than pure kinematics to judge them. What is encouraging is that there are several possible compositions to choose from which are guaranteed to meet the motion specifications.

We also measured the average time taken to generate a solution at each level in the search space. This is plotted in Figure 20. Note that the time to generate an eggbeater design grows very slowly with depth, whereas the time to generate a wiper design grows exponentially with depth. This phenomenon is explained by the solution density which shows that the number of compositions at a level that satisfy the wiper specifications are far fewer than the ones that satisfy the eggbeater specifications. These graphs suggests an interesting measure of the complexity of a design: a very hard design problem is one whose solution density (the ratio of the number of designs that satisfy the given specifications to the total number of designs at some depth) falls off rapidly with search depth; therefore the time to generate a solution at each search depth increases.

The third plot demonstrates how the average time to compute a solution varies with the number of primitive mechanisms. At a search depth of 4, we timed the design of a clock starting with 12

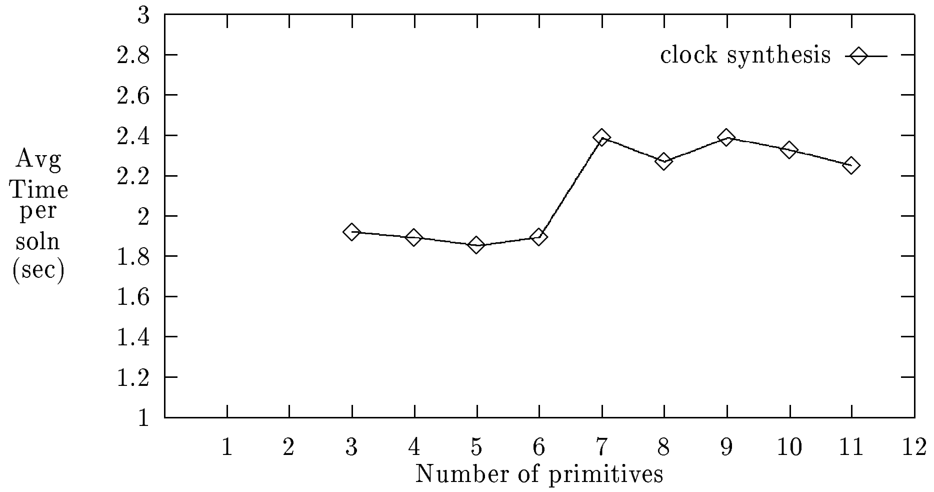


Figure 21: Performance Graph II

primitive mechanisms and by eliminating primitives one at a time. The plot reflects the fact that the clock design is dependent on a few of the primitives in the database and the times are not sensitive to the presence of other primitive mechanisms. Interestingly enough, our program fails quickly when there the set of primitive mechanisms is inadequate for a design.

7 Conclusions

This paper presented a case study of the integration of methods in qualitative physics and constraint programming with general algebraic reasoning on 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 behavioral 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($\mathcal{R}$) and all examples discussed in this paper are drawn from our implementation. Our base set of examples are drawn from mechanisms in [23] 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 [31] for synthesizing special classes of mechanisms e.g., cam-follower mechanisms, occupy an interesting middle ground between pre-parametric design schemes which requires high-level qualitative specifications and the numerical optimization packages which require very detailed kinematic specifications. Case-based methods [24, 28] 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. All the physical prototyping of the designs presented in this paper were performed using Technics Lego. We can integrate conceptual design systems through to detailed design and physical prototyping in a standard medium. This work may have implications for mechanical nano-technology designs because low-dimensional configuration spaces can be used to reason about shapes and motions in that domain. Computational scale issues will be studied with field work at the Xerox Webster Research Center.

Acknowledgments

We thank Roland Yap and Joxan Jaffar for help beyond the call of duty on CLP($\mathcal{R}$) Srikanth Kannapan and Brian Falkenhainer for comments on an early draft, the anonymous reviewers at Research in Engineering Design for their constructive suggestions which improved our exposition considerably, and Professor Freudenstein, Professor John Hopcroft, Dan Weld, Rick Palmer, and Dinesh Pai for help and encouragement throughout the research.

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] A. Chakrabarti. *Designing by functions.* PhD thesis, University of Cambridge, 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.*

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

[10] D. A. Hoeltzel and W.H. Chieng. Knowledge-base approaches to the creative synthesis of mechanisms. *Computer-aided design*, 22(1), 1990.

[11] S. Hoover and J. Rinderle. A synthesis strategy for mechanical devices. *Research in Engineering Design*, 1, 1989.

[12] L. Joskowicz. *Reasoning about Shape and Kinematic Function in Mechanical Devices.* PhD thesis, New York University, September 1988.

[13] L. Joskowicz and S. Addanki. From kinematics to shape: An approach to innovative design. In *Proceedings of AAAI-88*, pages 347–352. Morgan Kaufmann, 1988.

[14] L. Joskowicz and D. Neville. A representation language for mechanism design. Technical report, University of Washington, 1992. Also in the proceedings of QR 92.

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

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

[17] An algebraic and predicate logic approach to representation and reasoning in machine design. *Mechanism and Machine Theory*, 25(3):335–353, 1990.

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

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

[20] S. Kota and S. Chiou. Design representation and computational synthesis of mechanical motions. In D. L. Taylor and L. A. Stauffer, editors, *Design Theory and methodology 92*, volume 42, pages 365–372. ASME, 1992.

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

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

[23] D. Macaulay. *How Things Work*. Houghton Mifflin Co., 1990.

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

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

[26] D. Prabhu and D. Taylor. Synthesis of systems from specifications containing orientations and positions associated with generalized flow variables. Technical report, Cornell University, Mechanical Engineering Dept.

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

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

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

[30] 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.

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