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

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

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

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

[19] 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.

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

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

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

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

[9] 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.

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

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

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

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

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

output dynamical systems. His algorithmic focus is on optimizing these designs by function sharing. Finger and Rinderle [3] 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.

Our use of configuration spaces to study mechanism design has been inspired by the work of Joskowicz and Addanki [9] on synthesizing part shapes. While [2, 10] have used configuration space analyses for kinematic simulation, our method and its symbolic compilation are one of the first completely automated methods for conceptual kinematic synthesis.

There are other approaches to mechanism synthesis that can be profitably combined with our first-principles approach. Expert system techniques [19] for synthesizing special classes of mechanisms e.g., cam-follower mechanisms, occupy an interesting middle ground between the pre-parametric design scheme outlined here which requires high-level qualitative specifications and the numerical optimization packages which require very detailed kinematic specifications.

References

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

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

[3] 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.

[4] F. Freudenstein and L. Dobrjanskyj. On a theory of type synthesis of mechanisms. *ASME Transactions on Machines and Mechanisms.*

We now present a summary of relevant related work and place our contribution 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 [15, Chapter 11]. Pioneering work in the area of type and number synthesis was undertaken by Freudenstein [4] who proposed a generate-and-test method. The key idea in Freudenstein's work is to represent mechanisms using graph theory and to systematically enumerate mechanisms of various types with a given degree of freedom and a given number of links. The enumeration technique produces millions of mechanisms even for small DOFs and link numbers.

While Freudenstein's early generation work did not actively use the kinematic specification to guide the synthesis, his later work [5] took kinematic function into account. Kota, Kannapan and Ulrich devised goal-directed synthesis schemes based on motion, not function. Kota [13] proposed 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. Our abstraction scheme captures more detail and we have an analysis of when it works because it is grounded in a more detailed description language. Kannapan [11] developed another abstraction scheme for motion description geared for designs that involve shafts, handles, bearings and supports as primitives. Pieces of the primitives described in the language are combined together to form new compounds. He devised a system of transformation rules (rewrite rules for the motion description language) that systematically explored the space of possible combinations. Kannapan hand-simulated an interesting shaft coupling design using his method. Kannapan's techniques are not implemented and is little analysis of when the method works. Ulrich [18] also proposes a compositional technique for pre-parametric design. His design problems are specified qualitatively in the vocabulary of bond graphs [16] – effort and flow variables in electrical, mechanical and fluid media are the inputs and outputs. The abstract designs produced are bond graphs for single-input single-

sition 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 the 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 for our abstraction A to preserve joins. 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 [9] for synthesizing novel shapes into our design system to automatically extend our library of primitive abstract mechanisms.

7 Conclusions and Related Work

This paper presented a general method for the type and number synthesis of mechanisms from specifications of input and output motions grounded in a library of primitive motion relations. We then specialized the method to take qualitative constraints on motion to produce alternate 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. Future work involves extending the set of primitive relations, proving completeness properties for these relations, and integrating mechanism synthesis with multi-domain designs.

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

Table 2: Synthesizing single input, multiple output mechanisms

links of $\mathcal{M}_m^l$ and $\mathcal{M}_k^j$, as well as the output links of $\mathcal{M}_{k-1}^j$ are rigidly connected together. The optimization step merges identical motions, and therefore the final graph 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.

Theorem 3 *Algorithm* **SIMO_Synthesize** *is sound: i.e, it designs mechanisms that satisfy the qualitative 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 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 compo-

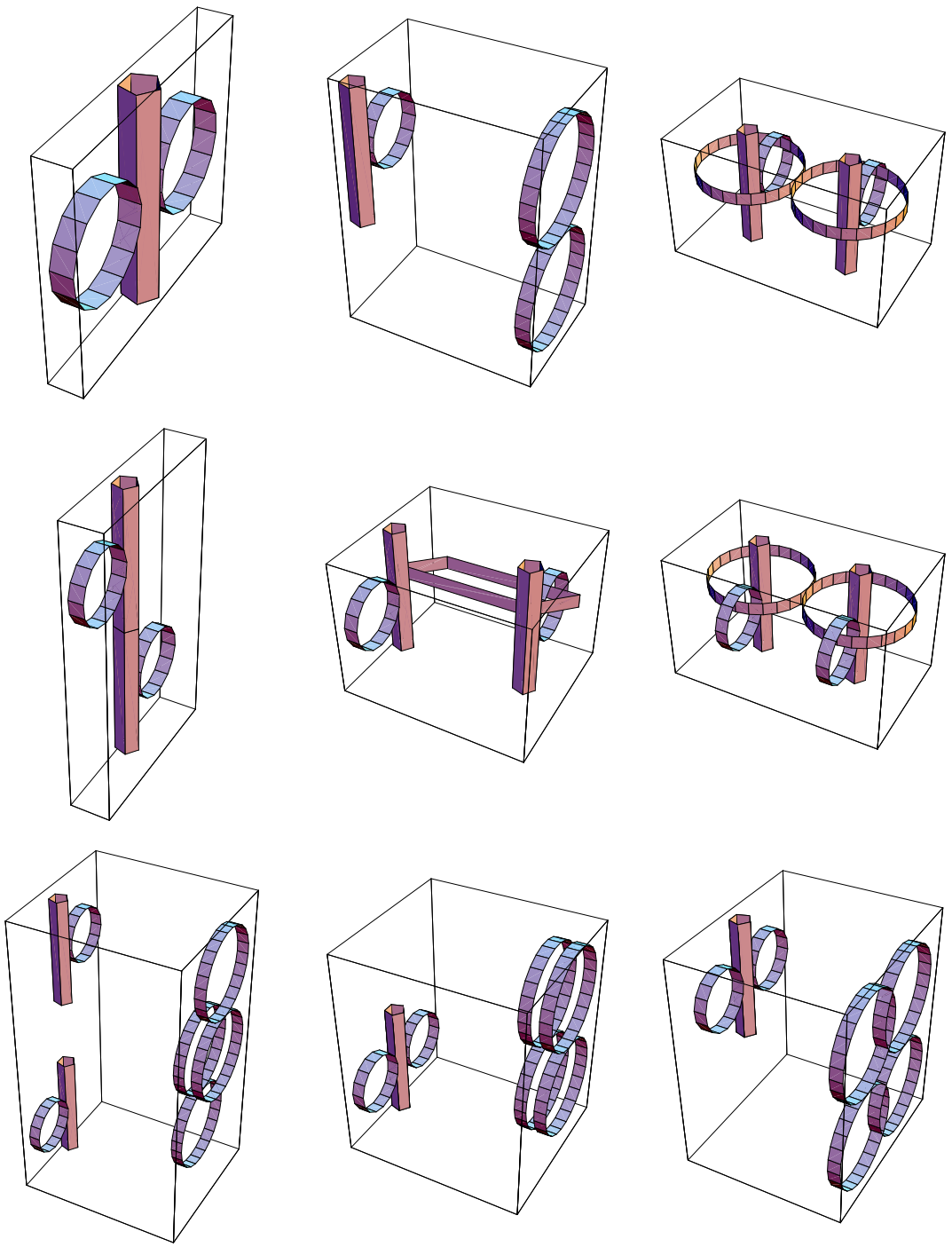

Figure 4: Conceptual Designs for an Eggbeater

the constraint accumulation process is a least-commitment strategy that minimizes backtracking in the space of compositions of primitive abstract mechanisms. For instance, 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 3 and its variants was also completed in about two seconds on a Sparc 1+.

6.2 Synthesizing single input, multiple output mechanisms

Many useful mechanisms produce multiple outputs from a single source. e.g., egg-beaters, cars. To design these single input multiple output (SIMO) mechanisms, we need to specify a sequence of output motions. Given Q_i, a qualitative specification of the input; $Q_o^1, \ldots, Q_o^n$, a sequence of output motions, and constraints on Q_i and Q_o's. Find a tree of mechanism descriptions, $\mathcal{A}_1^1, \ldots, \mathcal{A}_{k_1}^1, \mathcal{A}_1^2, \ldots, A_{k_n}^n$, which when composed satisfies the input-output specification, i.e.,

$$\forall 1 \leq j \leq n. \forall q_i \in Q_i. \exists q_o^j \in Q_o^j. (q_i, q_o^j) \in \mathcal{A}_1^j \bowtie \ldots \bowtie \mathcal{A}_{k_j}^j$$

The SIMO synthesis problem can be solved by a series of calls to **SISO_Synthesize**. Calls to **SISO_Synthesize** produce a tree with isolated paths from Q_i to each Q_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 the paths as follows: it looks for 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 such that $I_1 = I_2$, it eliminates the path from Q_i to I_2 and connect I_1 in place of I_2. This eliminates repeated transformation of Q_i to I_2. In the algorithm, we denote $\mathcal{M}$'s input motion by **InputMotionClass**($\mathcal{M}$). When we merge the inputs of $\mathcal{M}_m^l$ and $\mathcal{M}_k^j$ in the last step of **SIMO_Synthesize**, the input

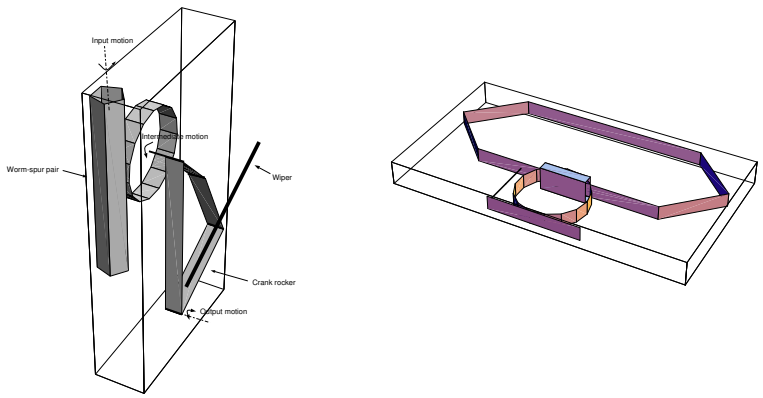

Figure 3: Design of a windshield wiper and a rack-and-pinion steering

Input motion Q_i:	**Output motion** Q_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 $0.5 < F < 2$, $\pi/2 < X < \pi$

The output produced by our program, displayed using Mathematica, is in Figure 3.

Theorem 2 *Algorithm* **SISO_Synthesize** *is sound: i.e, it designs 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. 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 new synthesis problem $i, T(i_p)$ is solved. When a linear primitive mechanism is chosen, we can calculate $T(i_p)$ very simply from i_p by the scaling factor that relates the input and output motions of the mechanism. If, however, the primitive mechanism is not linear, we use a general method to calculate i_p to $T(i_p)$.algebraic

```
synthesize(Input,Output,Plan,Depth) :-
        motion_eq(Input,Output).

%% linear mechanism
synthesize(Input,Output,[(Mechanism,Transform)|Plan],Depth) :-
        Depth > 0,
        mechanism(Mechanism,linear(Factor),NewInput,NewOutput),
        transformation(NewOutput,Output,Transform),
        linear_apply(Transform,Factor,NewInput,NewGoal),
        synthesize(Input,NewGoal,Plan,Depth-1).

%% nonlinear mechanism
synthesize(Input,Output,[(Mechanism,Transform)|Plan],Depth) :-
        Depth > 0,
        mechanism(Mechanism,nonlinear,NewInput,NewOutput),
        transformation(NewOutput,Output,Transform),
        nonlinear_apply(Transform,NewInput,NewGoal),
        synthesize(Input,NewGoal,Plan,Depth-1).
```

Consider the design of a windshield wiper whose input power is provided by a motor rapidly rotating around the z axis. The wiper oscillates in the yz plan with low frequency. We specify the problem as follows.

to some motion satisfying Q_o. To be exact, we want

$$\forall q_i \in Q_i . \exists q_o \in Q_o . (q_i, q_o) \in \mathcal{A}_1 \bowtie \ldots \bowtie \mathcal{A}_n$$

where $q_i \in Q_i$ means that the motion q_i is in the class of motions described by the qualitative specification Q_i.

6.1 Synthesizing single input, single output mechanisms

SISO_Synthesize(Q_i,Q_o)

1. If $Q_i = Q_o$, return the null machine.
2. Select an abstract mechanism $\mathcal{M}$ and a rigid transformation T such that

$$Q = \{q \mid (q, q_o) \in T(\mathcal{M}), q_o \in Q_o\} \neq \emptyset$$

   Thus Q is the (largest) set of motions that can be transformed by $T(\mathcal{M})$ to motions in Q_o. Q is the regression or backprojection of Q_o with respect to $T(\mathcal{M})$.

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

Table 1: Synthesizing single input, single output mechanisms

We implement the synthesis method as a depth-bounded, goal-directed, depth-first backward chainer in CLP($\mathcal{R}$). The operational model vof 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. The base case of the synthesis is 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. The recursive step first involves non-deterministic choice of a primitive mechanism (i_p, o_p). 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,

is atmost 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. A mechanism is 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 interval over which the output of a linear mechanism ranges is $aX + b$ where $a, b \in \mathbf{R}$ and X is the interval over which its input motion ranges. The frequency of the input motion is preserved by a linear mechanism.

The specific $\bowtie$ that we developed above is inadequate for handling non-linear mechanisms. To guarantee preservation of joins 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 in our framework, 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.

6 The Synthesis Algorithms

We reformulate the algebraic description of the abstract synthesis problem in terms of qualitative motion descriptions. The synthesis problem is: Given Q_i, a qualitative specification of the input motion; Q_o, a qualitative specification of the output motion, and constraints on Q_i and Q_o. Find a sequence of abstract mechanism descriptions which when composed by $\bowtie$ will transform any motion satisfying Q_i

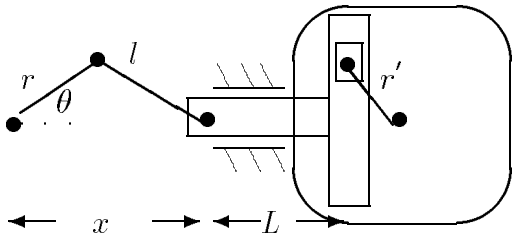

Figure 2: An example that shows join preservation violation

Definition 8 *The mapping A preserves joins if*

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

Note that this constraint places restrictions on the definition of $\bowtie$. For the simple language of rotations and rectilinear translations introduced above, $\bowtie$ consists of two sets of rules for unifying motion as in [8, 12] – one for rotations, the other for rectilinear translations. Two rotations can be unified (intersected, in C-space terms) if they have the same axes of rotation, their angular speeds and range are equal, and they are have the same frequency (if they are oscillations). This unification captures the join calculation in C-space and the abstract description filters exactly the properties of interest for composing the gear train example in Section 3. 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 join-preserving. To better understand this restriction, it is useful to consider the example in Figure 2 where join preservation is violated.

The output of the crank rocker in Figure 2 is a rectilinear translation (reciprocation) which is the input to the skotch-yoke mechanism. Our $\bowtie$ construction would unify the two motions as long as their ranges, speed, and axes coincided. Unfortunately, an analysis of the underlying C-space[2] relations reveals that there

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

5 Tractable Representations of 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 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 [9, 11, 13, 18] in its use of predicate calculus. However, unlike these approaches, but in common with [8], 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 join-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 which picks out specific properties of a motion, and necessarily omits other properties. 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 joins. If we can compute $A(\mathcal{R}_1 \bowtie \mathcal{R}_2)$ accurately and efficiently from $A(\mathcal{R}_1)$ and $A(\mathcal{R}_2)$, for configuration space motion relations $\mathcal{R}_1$ and $\mathcal{R}_2$, we have an adequate language. The formal property is called join-preservation and requires the specification of $\bowtie$ in the language specified by A.

Definition 7 *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 two-stage synthesis process, consisting of the abstract synthesis and the concrete synthesis phases, is sound. That is, if it produces a solution, it will satisfy the given motion constraints.*

We now discuss the computational complexity of the two phases of design. Let the cardinality of the set of primitive abstract mechanisms be n. In the abstract synthesis phase, the generation component can explore $\sum_{i=1}^{n} n^i$ compositions. Computing each composition involves intersection of algebraic sets: the worst case complexity is doubly exponential in the number of variables in the constraint set when a closed form solution is possible. 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.

4 The Synthesis Problem

We now cast the synthesis problem in the formal framework for motions and mechanisms that have 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 and these define a class of abstract mechanisms.

Definition 5 *The general synthesis problem is to find a concrete mechanism* (K, C)*, given constraints on an abstract mechanism* $\mathcal{R} \subseteq I \times O$*.*

Our compositional approach to solving the general synthesis problem grounds the syntheses in a set of primitive abstract mechanisms $\mathcal{P}$ 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 these abstract mechanisms. We will show that this method is sound: that is, it produces concrete mechanisms which implement the given motion specifications.

Definition 6 *The abstract synthesis problem is to find a sequence of abstract abstract mechanisms,* $\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_n$*, with known concrete implementations, where* $\mathcal{R}_1 \bowtie \mathcal{R}_2 \bowtie \ldots \bowtie \mathcal{R}_n$ *satisfies the given constraints on* $\mathcal{R}$*.*

A simple generate and test scheme for solving this problem repeatedly computes joins of elements in the set of primitive abstract mechanisms $\mathcal{P}$, and verifies that the composition satisfies the given constraint.

Concrete designs are produced by finding implementations for each of the primitive mechanisms that occur in the solution to the abstract synthesis problem. The primitive implementations are composed using Definition 4. This process involves simulateneously solving dimensional and geometric constraints from each of the chosen primitive implementations.

position using configuration space intersections suited for mechanism *simulation*. Joskowicz's composition operator, defined on subspaces of a configuration space, is absorptive, symmetric, commutative, associative and monotonic. Our serial composition operation, which is defined with *compositional synthesis* in mind, is associative and does not satisfy any of the other properties listed above.

An example of serial composition of two abstract mechanisms is illustrated in Figure 1B 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 $\psi' = -\alpha_2(-\alpha_1\theta + \phi) + \phi'$.

Definition 4 *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, provided $C_1 \cup C_2$ is a consistent set of constraints (has a solution).*

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

freedom, namely θ.

The definition of an abstract mechanism can be restated in terms of input and output motions. Let $\mathcal{R}$ be an abstract mechanism satisfying Definition 1, and let M_i and M_o be motions of the input and output links respectively. Then $\mathcal{R}$ can be defined as:

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

This describes all possible motion transformations of the mechanism and is sometimes more intuitive than Definition 1.

Definition 2 *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 motion specification.*

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 [6] annotated with a set of constraints.

3 Composition of Mechanisms

Complex mechanisms are composed out of simpler ones by serial composition.

Definition 3 *The serial 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*

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

Two configuration spaces are equal if and only if they denote the same set of configurations. To solve $o_1 = i_2$, we intersect the sets i_2 and o_1 to obtain the configurations common to them. Joskowicz [8] constructed a definition of com-

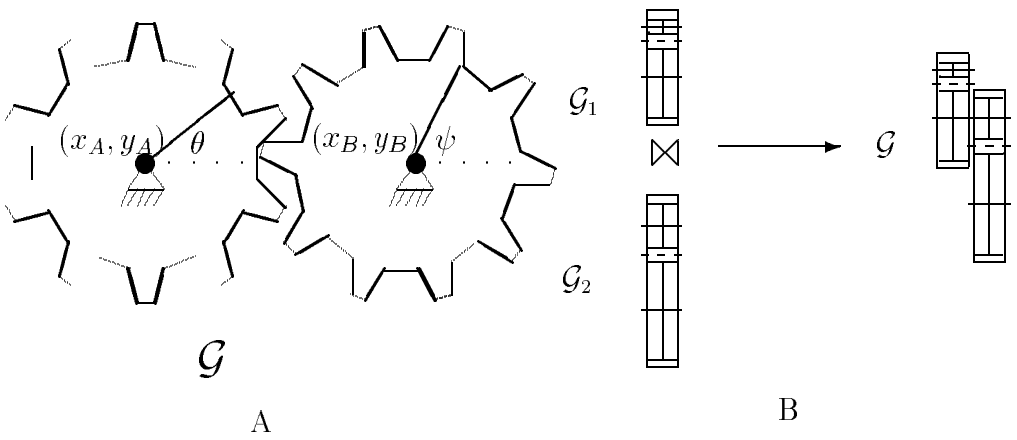

Figure 1: A meshed gear pair and a gear train composition

Definition 1 *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. Let d be the dimension of $\mathcal{R}$. $\mathcal{R}$ is a smooth manifold of dimension d.*

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) shown in Figure 1A. $\alpha > 0$ 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 $O = \{(x_B, y_B, \psi) \mid \psi \in [0, 2\pi)\}$,

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

This description[1] 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). The degree of freedom (DOF) of an abstract mechanism is its dimension. Note that $\mathcal{G}$ has only one degree of

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

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 general algebraic synthesis problem is presented in Section 4. In Section 5, we discuss tractable representations of the configuration space descriptions. We then present constraint-based algorithms that employ these representations. The algorithms have been implemented in $CLP(\mathcal{R})$ [7] and we present examples of interesting syntheses in Section 6. In Section 7 we present a summary of related work. We conclude by reiterating the main contributions of our paper and provide a discussion of future work on the problem of automating kinematic synthesis.

2 Configuration Spaces, Motions, and Mechanisms

The configuration of an object [14, pg. 8] is a specification of the position of every point on it relative to a fixed frame of reference. The configuration space of an object is the set of its possible configurations. The motion M of an object is a description of how its configuration changes with time [8]. The configuration of a planar rigid body undergoing pure rotation at time t, is $(x_A, y_A, \theta(t))$ where (x_A, y_A), the center of rotation, is a reference point on A. For uniform rotation with angular velocity ω, $\theta(t) = \omega t$.

Reuleaux [17] 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, and a *concrete mechanism* which is a 3D arrangement of rigid bodies that implements this relation.

1 Introduction

The broad goal of our research is to derive tractable computational theories of conceptual or pre-parametric design. In this paper, we focus on mechanisms and the problem of synthesizing them from descriptions of their input and output *motions*.

Conceptual synthesis is generally acknowledged to be a very difficult problem. A modern textbook in the area [15] 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 given kinematic constraints involves deriving geometry from motion: the inverse of the kinematic simulation problem. Conceptual design of mechanism is an active area of research [4, 9, 11, 12, 13, 19]. 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 constructible designs. Our approach allows us to exploit the best features of quantitative (algebraic and numeric) and qualitative (symbolic) motion description languages. At the finest level of detail, motion is formalized in terms of *configuration spaces* [14]. The motion of an object is viewed as a mapping from time to its configuration space. We develop a predicate vocabulary [10, 11, 12] that is an effective shorthand for some commonly occurring motions. We also develop motion unification rules that compile out a specific computation on configuration spaces. The predicate language permits high-level reasoning with certain classes of motions and the design of efficient synthesis procedures. We then specify the class of synthesis problems for which the abstraction sanctioned by the predicate descriptions yield correct designs in a computationally effective manner.

Constraint-Based Kinematic Synthesis*

Devika Subramanian† Cheuk-San (Edward) Wang‡

Abstract

This paper presents a constraint-satisfaction algorithm for kinematic synthesis of mechanisms with fixed topology. A mechanism is defined as a relation $\mathcal{R} : I \times O$ on the configuration space I of its input links and the configuration space O of its output links. The algorithm is based on a composition operation on configuration spaces. It is a symbolic compilation of a general algebraic technique for synthesis of rigid-part mechanisms. The class of synthesis problems and motion description languages that the algorithm can solve is identified. The correctness of the algorithm is shown using its configuration space foundations. Experimental results are obtained with a prototype system implemented in $CLP(\mathcal{R})$ [7].

Area: Principles of AI applications, Qualitative Reasoning.

Length: 4400 words.

Slightly revised version of a paper submitted to IJCAI-93.

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

†Department of Computer Science, Cornell University, Ithaca, NY 14853, devika@cs.cornell.edu, (607) 255-9189

‡Department of Computer Science, Cornell University, Ithaca, NY 14853, wang@cs.cornell.edu, (607) 255-5565