Conceptual Synthesis of Mechanisms from Qualitative Specifications of Behaviour*

Devika Subramanian
Cheuk-San Wang, Scott Stoller, Arjun Kapur

Computer Science Department
Cornell University
Ithaca, New York 14853

13 January 1992

Abstract

This paper describes a novel method for the synthesis of mechanisms from their qualitative behavioural specifications. The technique has been implemented in the context of automated design of mechanisms in Technics Lego. We introduce a new kinematic abstraction called a *unity machine* which is a machine with a single degree of freedom. Unity machines are the mechanical analogues of functions in programming languages. Unity machines are the key abstraction that allow for compositional synthesis of mechanisms from a basis set, guided by functional specifications. Geometric constraints accumulated during unity machine synthesis are satisfied by a qualitative kinematics analyzer based on Kramer's [Kramer] geometric engine. This allows us to cleanly integrate detailed dimensional synthesis by feature-oriented refinement with the abstract unity machine synthesis. Three steps in the design process – unity machine construction, qualitative kinematic analysis, and refinement – are illustrated in the context of the design of an autonomous tripod walker.

*This work was supported by NSF-IRI-8907271 and the Xerox Design Research Institute. The Legoites are listed in reverse alphabetical order. The first Legoites who worked on an early version of the Lego synthesis program were Scott Benson and Robert Wisniewski. Their help and participation in the Cornell Design Project is gratefully acknowledged.

1 Introduction

The design problem is the calculation of the structure of the desired artifact in some implementation medium, from a specification of its function or behaviour[1]. The keys to automated conceptual synthesis are:1) the design of an effective language for specification of structure and behaviour 2) the design of effective abstractions that bridge behavioural and structural descriptions. This paper presents a simple specification language, and a new kinematic abstraction called *unity machines* that links structure and behaviour.

Our main goal is to design new qualitative representations suited for kinematic synthesis. Kinematic synthesis is our focus because there are no algorithmic methods known for synthesizing mechanisms from descriptions of kinematic behaviour. The standard process of designing mechanisms consists of selecting a mechanism type from a catalog [Artobolevsky], and then calculating the actual dimensions of the links using numerical methods. Engineers typically rely on intuition and experience for selection of an initial conceptual design from the catalog. In this paper, we demonstrate the power of qualitative representations for automating conceptual design of mechanisms, and the clean integration of qualitative and quantitative methods for completing detailed design.

Our allied goal is to provide power tools for designers for creating new conceptual designs of devices in multiple implementation media using behavioral specifications. We illustrate the ideas in this paper in the context of the design of a 6-legged tripod walker which can move forward and backward at a rate of 3 meters/minute. The implementation medium is the Technics Lego set, which is a universal set which can realize all the lower kinematic pairs and several important higher pairs [Reuleaux]. Our behavioural specification language allows for both qualitative and quantitative information. Qualitative information about the type of motion is used to guide the conceptual synthesis. Quantitative information is used during the refinement to a concrete design.

- Given: an input shaft oriented along the x axis, uniformly rotating at 1200 rpm.
- Find: a device with 6 outputs each of which traces a closed elliptical path parallel to the x-z plane and where all paths are traced in the same sense (clockwise or counterclockwise). Additionally, outputs 1,3,5, are in phase with respect to each other, and are each π out of phase with respect to the in-phase outputs 2,4 and 6. The six outputs are physically separated in space such that at any instant the endpoints of output links 1,3 and 5 as well as those of 2,4, and 6 form non-degenerate triangles parallel to the xy plane.

The Technics Lego world is an implementation domain with low setup cost, non-trivial geometry which is a challenge for most present day three-dimensional modelers, and with which a combinatorial number of potential assemblies can be built. A kinematic pair can be realized in an average of about 4 ways in the set, so a typical assembly with 20 kinematic pairs can be put together in about 4^{20} ways. The need for abstract descriptions and their subsequent refinement is essential for containing the complexity of synthesis from function in this domain. Our design method, shown in Figure 4, consists of the four steps outlined below.

1. The construction of a composite unity machine from the input-output specification.
2. The satisfaction of the geometric constraints accumulated during the previous step by qualitative kinematic analysis [Kramer].

[1]To be very precise, we will use information about behaviour only. There is considerable confusion in the literature about the use of the terms function and behaviour. For us, function relates to how the mechanism interacts with the environment and behaviour relates to intrinsic motion descriptions.

3. Feature-oriented refinement of the unity machine to a concrete Lego design. This can be preceded by a pre-processing step which minimizes the machine by exploiting function sharing [Ulrich] using constraints imposed by the physical realization in the given implementation medium.

4. Quantitative kinematic analysis of the refined design.

The key kinematic abstraction employed is the use of machines with a single degree of freedom called *unity machines*. Unity machines can be composed like mathematical functions. Each unity machine is viewed as a motion transformer. The decomposition of the design problem via unity machines reduces the complexity of the problem as compared to a pure piece level synthesis. This paper focuses on kinematics alone; however, Figure 4 shows how information about dynamics can be incorporated in using bond graphs [Karnopp]. In the rest of the article, we will step through the first three phases described above in the context of the design of the tripod walker.

2 The Design Method

This section describes the synthesis method which produces a conceptual design for a mechanism. The next section explains the refinement process that leads to a concrete design. We first introduce the formal language used for specifying inputs and outputs to our synthesis system.

2.1 Describing Behaviour

We use first-order predicate calculus to describe the various properties of input and output motion. Motions are objects in our language. We introduce the predicates in the language by describing the input and outputs of our tripod walker.
`input(walker,i), type(i,rotation), raxis(i,x-axis),`
`speed(i,.05), sense(i,clockwise), continuous(i).`
The sentence above describes the input motion i to be a continuous, clockwise rotation about the x-axis with an angular speed of 0.05 (1200 rpm). Commas denote logical conjunction. The language allows for the specification of both qualitative and quantitative properties. The 6 outputs of the walker are described as follows: for $1 \leq k \leq 6$:
output(walker,o_k), path(o_k,ellipse), speed(o_k,P),
P < .005, sense(o_k,clockwise), continuous(o_k)
The `path` predicate allows us to specify the curve traced by a translational motion. The phase relationships between the motions are described as follows:
phase(o_i,o_j,0) where i and j are of the same parity (even or odd), $1 \leq i, j \leq 6$,
phase(o_i,o_j,π) where i and j are of different parity, $1 \leq i, j \leq 6$

We can also describe geometric constraints between motions. We can state that two rotational axes are parallel. More predicates in our language are introduced by their use in the later sections.
`type(i,rotation), type(j,rotation), r-axis(i,A), r-axis(j,B), parallel(A,B)`

The complete set of predicates for describing motion are in Figure 1. One might ask whether our choice of vocabulary could be justified by a principled analysis. These motion descriptors appear to be adequate for describing a large number of the mechanisms in Artobolevsky catalog[Artobolevsky] of mechanisms and are in fact used in the informal explanations that accompany the figures. Clearly, this vocabulary defines a finite number of equivalence classes of motion and sets up the stage for defining an orthogonal set of basis unity machines that convert one motion class to another.

`type(M,X)`	:	motion M is of type X $\in${rotation, translation, rotation+translation}
`sense(M,X)`	:	applies to rotations only and X $\in$ {clockwise, counterclockwise}
`reverse-dir(M)`	:	motion M reverses sense (oscillation or reciprocation)
`period(M,X)`	:	periodic motion M where X is the time taken to complete one cycle
`continuous(M)`	:	motion M is continuous (has zero dwell time)
`intermittent(M)`	:	motion M has non-zero dwell time
`raxis(M,X)`	:	rotation M has axis A (a directed line in space)
`tpath(M,X)`	:	motion M traces path of geometric description X $\in$ {line, circle, ellipse, helix, screw}.
`phase(M,N,P)`	:	motion M has phase P relative to motion N
`range(M,R)`	:	motion M has range R
`speed(M,S)`	:	motion M has speed S

Figure 1: The motion description vocabulary

2.2 Describing Structure

A key abstraction called a *Freudenstein diagram* [Freudenstein74], or F-diagram for short, captures those aspects of mechanisms that are determined solely by kinematic pairing and are independent of metric properties[2]. An example of such a property is the number of degrees of freedom (DOF) possessed by a mechanism. An unconstrained object in space has three translational and three rotational degrees of freedom. When it is kinematically paired with another object, the first object loses at least one degree of freedom. F-diagrams are introduced in most textbooks on mechanisms [MabieRein].

Definition 1 *The F-diagram of a mechanism is an undirected graph with vertex set V and edge set E such that every link in the mechanism is represented by a vertex in V, and for every kinematic pair between two links in the mechanism, there is a labeled edge in the F-diagram between the vertices that denote the links. There are some vertices called* ground *vertices which stand for the links which are grounded.*

Edges are labeled by the type of kinematic pair that exists between the two links. Our label set consists of the six lower pairs (revolute, prismatic, helical, spherical, planar, and cylindrical) and four higher pairs (gear pairs, worm gear, rack-and-pinion, and cams).

The F-diagram corresponding to a pair of meshed gears is in Figure 5. An F-diagram of a mechanism is an abstract representation because dimensional information is lost in the construction. Thus the diagram in Figure 5 could well represent a planetary gear system where a simple gear rotates along the toothed inner circumference of a larger gear. F-diagrams can be used to compute the mobility, or the total number of degrees of freedom of a mechanism. For non-degenerate *planar* mechanisms with n links of which g links are ground, and f_1 one-degree-of-freedom (1-DOF) joints and f_2 2-DOF joints, we can determine the mobility M, using Grubler's equation [3].

$$M = 3(n-g) - 2f_1 - f_2$$

Plane revolute and plane prismatic joints both are 1-DOF joints.

[2]Our formalism allows for introduction of metric properties back into the abstraction.

[3]For non-degenerate *spatial* mechanisms with n links of which g links are ground, the mobility is determined by Kutzbach's equation.

$$M = 6(n-g) - 5f_1 - 4f_2 - 3f_3 - 2f_4 - f_5$$

where f_i is the number of i-DOF joints for $1 \leq i \leq 5$.

2.3 Unity Machines: Linking Structure and Behaviour

Definition 2 *A special class of F-diagrams which denote mechanisms with exactly one degree of freedom are called* unity machines. *Unity machines have a distinguished vertex called the* input *which is the input link of the mechanism, several* output *vertices which denote the output links of the mechanism, and one or more ground links.*

The mechanism `um1` in Figure 5 is a unity machine. Its mobility is determined from Grubler's equation as follows. The gear pair is a 2-DOF joint. A revolute joint is a 1-DOF joint.

$$M = 3(3-1) - 2 \cdot 2 - 1 = 1$$

Node 1 of `um1` is the input vertex and Node 2, the output vertex. The ground link of the mechanism is denoted by a special grounding symbol in the figure. Neither the input nor the output link of a unity machine can be grounded.

Unity machines are transformers of motion. We annotate the input and output vertices of a unity machine by descriptions of motion. For instance, the gear pair `um1` in Figure 5 has a pure rotational input and converts it to a pure rotational output of the opposite sense (clockwise to counterclockwise, or vice-versa). We characterize the relation between input and output motions using behavioural rules. The behavioural rules for `um1` are shown below. Constants are denoted by lower-case letters and variables are upper-case letters. The symbol ⇒ denotes logical implication. The speed ratio `r` in the rules is simply the ratio of the number of teeth in the gear pair that implements the edge labeled G in `um1`.

1. `input(um1,I), type(I,rotation), raxis(I,A), sense(I,C), speed(I,S), speed-ratio(um1,r), output(um1,J)` ⇒ `type(J,rotation), raxis(J,B), sense(J, not C), speed(J,r · S), parallel(A,B).`
2. `input(um1,I), continuous(I), output(um1,J)` ⇒ `continuous(J).`
3. `input(um1,I), intermittent(I), output(um1,J)` ⇒ `intermittent(J).`
4. `input(um1,I), period(I,P), output(um1,J)` ⇒ `period(J,P).`

The input and output of `um1` are rotations about different centers. The axes of rotation are parallel to each other. The first rule describes the change to the input motion made by the machine: the speed is changed by the ratio `r` and the sense of rotation is reversed. `um1` preserves the continuity and periodicity of the input motion. If the input motion is intermittent, the output motion is also intermittent. Properties of motion preserved by `um1` are expressed by rules 2, 3 and 4 which are *frame axioms.*

A more interesting unity machine is shown in Figure 6. This machine is an inversion of a slider crank and forms part of the leg assembly of the walker. The behaviour of this machine is captured by the following rules. The input to the machine is a pure rotation, its output is a path traced by a specific point on the output link. Note that `um2` preserves the continuity and periodicity of the input motion.

1. `input(um2,I), type(I,rotation), raxis(I,A), sense(I,C), speed(I,S), output(um2,J)` ⇒ `type(J,translation), tpath(J,ellipse), speed(J,S), sense(J,not C).`
2. `input(um2,I), continuous(I), output(um2,J)` ⇒ `continuous(J).`
3. `input(um2,I), intermittent(I), output(um2,J)` ⇒ `intermittent(J).`
4. `input(um2,I), period(I,P), output(um2,J)` ⇒ `period(J,P).`

2.4 The Elementary Unity Machines

The equivalence classes of motion described by our vocabulary define various classes of rotations, translations and mixed motions. The rotations that are distinguishable are listed below.

1. `type(M,rotation), continuous(M), sense(M,S)`
2. `type(M,rotation), continuous(M), reverse-dir(M)`
3. `type(M,rotation), intermittent(M), sense(M,S)`
4. `type(M,rotation), intermittent(M), reverse-dir(M)`

Every rotation has an associated speed, and axis of rotation. Two rotations can be related by a phase described by the `phase` predicate. Rotations have sense: clockwise and counterclockwise. The `reverse-dir` predicate when applied to rotation describes oscillatory motion. The angular range of oscillation is captured by the predicate `range`. Note that each entry in the table stands for an infinite set of rotary motions over all speeds, axes of rotation, and ranges of motion.

The types of translational motions that can be distinguished include the ones in the table below. Translational motions trace a path described by the predicate `tpath(M,X)` where `X` can be a line, circle, ellipse, helix or screw. A line is represented by a pair consisting of a point on the line in 3-space and a vector that describes its orientation. A circle is described by a triple (radius, normal-vector, center) where radius is a real number, normal-vector is a unit vector perpendicular to the plane of the circle, and center is the location of the center of the circle in 3-space. Similar geometric descriptions are given for the other objects.

5. `type(M,translation), continuous(M), tpath(M,X)`
6. `type(M,translation), intermittent(M), tpath(M,X)`
7. `type(M,translation), continuous(M), tpath(M,X), reverse-dir(M)`
8. `type(M,translation), intermittent(M), tpath(M,X), reverse-dir(M)`

We have identified 26 equivalence classes of motion: 4 translation types each with 5 possible paths that can be traced, 2 rotations with definite sense each with two possible values for sense, and 2 oscillatory rotations. Note that each class contains motions with speeds, phases and range drawn from the set of real numbers.

We call these *basic motions*. We can now provide a basis set of elementary unity machines that convert one basic motion class to another. This allows us to obtain a complete set of unity machines relative to our motion description vocabulary.

Definition 3 *An elementary unity machine is a unity machine that cannot be composed from other unity machines. It converts one basic motion in our system to another basic motion.*

Elementary unity machines are thus the atoms of our motion conversion system. Some elements of the basis set of unity machines are in Figure 7: these are all the possible conversions from rotary motion. We represent the elementary motion converters as F-diagrams. The common names of the mechanisms and their index number in Artebelovsky's catalog are shown in the Figure. We have a database of elementary unity machines indexed by their input and output motions to facilitate queries of the form: list the unity machines which have inputs matching X, as well as list the unity machines with outputs that unify with Y.

2.5 Composing Unity Machines

The power of unity machines as the building block of our synthesis system comes from the observation that the composition of two unity machines yields another unity machine. Thus complex unity machines can be built from simpler ones by a compositional process. What needs to be explained now is the nature of the composition that preserves the single DOF property.

The composition of unity machines is achieved by merging the output link of a unity machine with the input link of another, provided that their motions are unifiable. That is, the set intersection of the two motions should not be the empty set. Since a finite number of motion types can be distinguished in our vocabulary of motion, we can specify the unification rules exhaustively.

A simple example serves to illustrate the concept of motion unification. Consider the unity machine `um1` introduced before. We will compose `um1` with a copy of itself, `um1`$'$, to produce a new unity machine. `um1` has input link 1 and output link 3 (link 2 is ground). `um1`$'$ with input link 4, output link 6 and ground link 5 is shown in Figure 9. We will unify the motions of link 3 and link 4.

`output(um1,M),`
`type(M,rotation),`
`continuous(M),`
`sense(M,not C),`
`raxis(M,A)`
	`input(um1',N),`
	`type(N,rotation),`
	`continuous(N),`
	`sense(N,W),`
	`raxis(N,B)`

Motion M and N can be unified because they are both continuous rotations. However, we must ensure that they have the same sense and that they have the same rotational axis. We specify the latter constraint by using the `associate-marker(marker,motion,location)` predicate, which associates a symbol `marker` with `motion` at `location`. A marker is simply a local coordinate frame attached to a joint on the F-diagram. A marker's global orientation and position are provided as a unit vector which identifies the direction of the z axis of the local coordinate frame, and a point identifying the origin of the marker. Thus, unification of `M` and `N` yields the constraints given below, where `J1` and `J2` are the locations of these rotations.

`not C = W` (rotations' senses should be equal)
`associate-marker(M1,M,J1), associate-marker(M2,N,J2), inline(M1,M2)`

These geometric constraints are checked for feasibility in the qualitative kinematic simulation phase of the design.

Intermittent rotations unify with intermittent rotations. The unification constraints are that they share the same sense, the same rotational axis, the same angular range and the same period. Continuous translations unify with continuous translations. The path objects traced by two unifiable translations have to be unifiable also. Thus if we have two continuous translations that trace line objects `tpath(M,lineA)` and `tpath(N,lineB)`, our unifier generates the geometric constraint that the lines are parallel.

2.6 The Synthesis Algorithm

The algorithm [Nilsson] used for the synthesis is a backward chainer that starts from the goal motion desired and composes a unity machine that matches the specification. The algorithm shown in Figure 2 handles single-input, single-output unity machine synthesis. It is a non-deterministic procedure as written: step 2 involves picking some elementary unity machine in the database whose output unifies with the required output motion. This procedure is sound: a compound unity machine produced by it is guaranteed to satisfy the input output relation. The procedure is also complete: if there exists a unity machine that satisfies the input output relation and which is composable from the elementary unity machines, the procedure will find it.

The synthesis algorithm produces the design shown in Figure 10 given the problem of synthesizing one of the legs of the walker. The geometric constraints produced, which are omitted from

procedure Single-Synthesize(Input,Output)

Given : Input: specification of the input and
Output: specification of the output.

Find : A sequence of elementary unity machines which when composed satisfy the input-output specification, along with a set of geometric constraints.

1. If Output unifies with Input return the null unity machine, the empty substitution, and the empty constraint set.
2. Let U(I,O) be an elementary unity machine whose output O unifies with Output with substitution θ. Let $I\theta$ be the substitution instance of the input of this elementary unity machine. Let C denote the set of geometric unification constraints generated.
3. Let A, θ', C$'$ be the unity machine sequence, substitution, constraints returned by Synthesize(Input,$I\theta$).
4. Return append(A,U(I,O)), $\theta' \circ \theta$, and C $\cup$ C$'$.

Figure 2: Synthesizing Single Input, Single Output Unity Machines

the figure for clarity, will be satisfied during the kinematic simulation stage described in the next section.

2.7 Optimization of Unity Machines by Function Sharing

The synthesis procedure given in the previous subsection handles one output at a time. To synthesize the 6-legged walker we need to achieve 6 subgoals that describe the motions of the individual legs, as well as the phase relationships between these motions. A synthesis procedure that satisfies multiple goals in sequence is described in Figure 3. The first phase of the multiple goal synthesis procedure produces modular designs, where no functionality is shared. The six subgoals for the legged walker are achieved in sequence resulting in six copies of the unity machine shown in Figure 10. Our multi-output synthesis procedure works very much like a modern day compiler: the first phase generates an intermediate design which is then optimized in the second phase. The optimization looks for opportunities to merge kinematic function: it identifies common functionality achieved at different points in the design, which can be achieved by a single unity machine. The idea is to perform *common-subexpression-elimination* on the initial design. Algorithmically, it involves detecting common subgraphs in the F-diagram and replacing them by a single instance.

In the case of the design of the walker, we could share the initial step-down of rotation obtained by unity machine `um3`. The subgraph that stands for `um3` in all 6 legs can be replaced by a single copy of `um3` whose output is fanned out. An even better design would share the functionality of both `um1` and `um3`: however the need to satisfy the phase relations between the motions (which is achieved by `um1`) invalidates this attempted optimization step. The algorithm in Figure 3 only searches for individual elementary unity machines that are shared among a set of unity machines. This optimization step has low complexity. The detection of larger common subgraphs (i.e., compositions of elementary unity machines) makes the optimization problem NP-complete. The application of the optimisation algorithm to the synthesis of the walker produces the design shown in Figure 11.

procedure Synthesize(Input,Outputs)

Given : Input: specification of the input and
Outputs: a sequence $\text{Output}_1 \ldots \text{Output}_n$ of motions to be synthesized.

Find : A sequence of elementary unity machines which when composed
satisfy the input-output specification,
along with a set C (initially empty) of geometric constraints.

```
For i from 1 to n do
    um_i, C_i <— single-synthesize(Input,Output_i)


For i from 1 to n do
    For j from 1 to n do
        for each u_k in um_i do
            for each v_l in um_j do
                if i ≠ j then if Input(u_k) = Input(v_l) and Output(u_k) = Output(v_l)
                              then merge u_k and v_l.
```

Figure 3: Synthesizing Single Input, Multiple Output Unity Machines

2.8 Analyzing Complex Devices as Finite State Machines

Unity machines are a useful abstraction for understanding the construction of complex mechanisms from simpler ones. Unity machines are to kinematics what side-effect-free programs are to functional languages. They allow for simple compositional synthesis, and provide opportunities for further optimization by function sharing.

Unity machines can be used to describe devices with more than one degree of freedom provided they can be discretized into machines with one degree of freedom. This is akin to representing an output function $f(x, y)$ as the union of one-argument functions $\bigcup_{x_i \in X} f(x_i, y)$ where X is the set of allowed values of the first argument. This strategy of decomposition works as long as X is a finite set. A two speed gear box can be thought of as a 2 state unity machine. In one state, it is a unity machine with a low gear ratio; in the other, it is a unity machine with a high gear ratio. The transition between the two states is determined by the position of the gear-shift lever. Methods of synthesizing finite state automata from high-level specifications can be brought to bear on synthesizing machines with more than one degree of freedom. This is currently being explored.

3 Kinematic Simulation of Lego Mechanisms

Kinematics is the study of position, velocity and acceleration of links without considering the forces that produce them. A kinematic analysis produces the velocity and acceleration of all the links in a mechanism given a description of the input motion. It can also generate the path traced by the mechanism for given driving inputs. A good way to view kinematics is as geometry in motion [Shigley].

Our project uses simulation in two ways: to satisfy geometric constraints generated during unity machine synthesis, and to animate a completed mechanism. For animation, the analysis must be quantitative and exact; for geometric constraint satisfaction during synthesis, qualitative simulation suffices because the designs are abstract during this phase.

Most texts on machines and mechanisms [Shigley, MabieRein] provide excellent numerical methods for detailed kinematic analysis. Commercially available programs like ADAMS use iterative techniques like the Newton-Raphson method with efficient encodings of the geometric constraints [Orlandea]. A numerical simulation requires a complete geometric description of the mechanism, whereas qualitative simulation only requires the information present in unity machines. The output of the qualitative simulator is a set of symbolic translation and rotation operations [Kramer] that *assemble* the mechanism so that the geometric constraints are satisfied. When the dimensions of the links of the mechanisms are instantiated, we obtain an exact quantitative simulation that can be used for animation of the mechanism.

3.1 Qualitative Simulation

The kinematic simulation method described here satisfies the geometric constraints accumulated by the synthesis algorithm. We employ a technique called degrees of freedom (DOF) analysis devised by Kramer [Kramer] which produces a symbolic solution to the positions of the various links in the unity machine. Kramer considered lower kinematic pairs only; we have extended his method to handle higher kinematic pairs (gears). We present a short summary of the method.

3.2 Degrees of Freedom Analysis

DOF considers the mechanism, described as an F-diagram, as an unassembled collection of links, initially. Each kinematic pair in the F-diagram is translated into a set of geometric constraints about the relative positions and orientations of the links. The geometric constraints accumulated during synthesis are added to this set. The constraints are then solved incrementally. The result of the DOF analysis is a metaphorical assembly plan (MAP), which is a sequence of translations and rotations of the links that assembles the mechanism for given values of the driving inputs. DOF analysis is well suited for producing animation since, in analyzing a mechanism, it generates a symbolic MAP once, which can be executed efficiently for various input values.

3.2.1 Markers and Geometric Constraints

We now show the algorithm that generates MAPs in the context of the 4-bar linkage. A critical concept in the algorithm is a *marker*, introduced earlier. The marker's global location and orientation are denoted `gmp(M)` and `gmz(M)` respectively. The latter stands for the global orientation of the z-axis of the marker. Kramer uses a set of primitive geometric constraints between markers. These are

`coincident`(M_1, M_2)	:	markers M_1 and M_2 have the same location in space
`inline`(M_1, M_2)	:	M_1 lies on the line through M_2 parallel to M_2's z-axis.
`in-plane`(M_1, M_2)	:	M_1 lies in the plane through M_2 normal to M_2's z-axis.
`parallel-z`(M_1, M_2)	:	the z-axes of M_1 and M_2 are parallel.
`rigid`(M_1, M_2)	:	relative motion between M_1 and M_2 is not possible.

Geometric relationships among the links in the mechanism are described in terms of relationships between markers. For the four bar linkage in Figure 12, we attach markers D_1 and D_2 to the left and right ends of the ground link (r_1), markers A_1 and A_2 to the bottom and top ends of the input link (r_2), markers B_1 and B_2 to the left and right ends of the connecting rod (r_3) and markers C_1 and C_2 to the top and bottom ends of the output link (r_4). We can now describe the connections between the links as: `revolute`(D_1,A_1),`revolute`(C_2,D_2), `revolute`(A_2,B_1) `revolute`(B_2,C_1), `angle`(D_1, A_1, θ_2)

A revolute joint can be defined in terms of the primitive set of constraints.
`revolute`(M_1,M_2) = `coincident`(M_1,M_2) $\wedge$ `parallel-z`(M_1,M_2)

3.2.2 The Construction of MAPs

In the construction of the metaphorical assembly plan for the four bar linkage, constraints are satisfied one at a time by moving objects by the actions of translation and rotation. Constraints, once satisfied, are maintained as invariants for the remainder of the analysis. Initially each link has 3 rotational and 3 translational degrees of freedom. The MAP produced by this analyis for the four-bar linkage problem is :

1. Translate link r_1 (resp. r_4) so that one of its ends is coincident with the point D_1 (resp. D_2) on the ground.

2. Rotate link r_1 about D_1 to satisfy the input angle constraint θ_2.

3. Determine the intersections of the loci of endpoints of links r_1 and r_4 and pick one of intersection points, say I (assuming one exists).

4. Rotate r_4 about D_2 so its endpoint is coincident with I.

5. Translate and rotate r_3 so its endpoints coincide with those of r_2 and r_4.

The actions that can achieve a constraint while maintaining others, are stored in a structure called the Plan Fragment Table (PFT). Entries in the table are indexed by the type of constraint that the plan satisfies. All of the entries in the PFT require that one of the markers involved in a constraint be fixed to a ground link. The plan fragment describes how the constraint is satisfied by translating and/or rotating the movable link. The kinematic analyzer examines the constraints involving the fixed markers first and selects the plan fragments which will achieve them. The entire PFT can be found in [Kramer]. To solve for the constraints on the floating link, locus analysis is employed. The locus of possible locations for B_1 are constrained by that of A_2, with which it is coincident to. A_2 has one rotational degree of freedom. Similarly, the locus of possible locations for B_2 are constrained by that of C_1, which also has one rotational degree of freedom. Locus analysis intersects these two sets to obtain solutions for the position of the middle link. Locus tables have the geometric description of the locus of links like A and C which have one rotational degree of freedom. Once the intersection points are found, the positions of the markers B_1 and B_2 are known, and link B can be positioned by appropriate translations and rotations.

A limitation of Kramer's DOF analysis technique is that it handles only a restricted set of joints; namely, the six lower pairs and a few higher pairs (universal joint and slotted-pin joint). Prominent among the higher pairs that it does not handle are toothed joints, such as connections involving gears, racks, and worm gears. Since such joints play a key role in many mechanisms, we have extended the method to handle them. We use a preprocessing step outlined in the appendix to convert gear-gear connections into assertions involving only the primitive geometric constraints defined by Kramer, since adding additional constraints could significantly complicate the analysis.

The DOF analyzer has been partially implemented; we are currently working on completing the implementation. As a test of the existing code, we simulated the motion of one of the walker's legs. The leg is described abstractly by the F-diagram in Figure 7. The dimensions of the links were obtained by instantiating the mechanism using Lego pieces. The section of the analyzer that constructs plans is implemented including the augmentations to include analysis of gears. The plan was executed for various values of the driving input, producing for each value the positions and orientations of all of the pieces. This output was used to create an animated sequence showing the walker in motion[4].

4 Refinement

Given an F-diagram of the unity machine to be synthesized, we first convert it to a more convenient line-graph where the edges stand for links and where the nodes stand for the joints in the mechanism.

Definition 4 *The* line graph *D of an F-diagram F is a graph whose edge set is the vertex set of F and whose vertex set is the edge set of F. There is an edge labeled j from node x to node y in D if and only if x and y meet at a vertex j in F.*

The line graph of the simplest unity machine `um1` is shown in Figure 13. The refinement problem is: given a line graph for a mechanism, to find sets of interacting Lego components that implement the edges so that they form the joints represented by the vertices of the graph. Our basic approach to this problem is to use *feature-oriented* refinement. An example illustrates the idea: there are exactly two ways to create s(see the transcript file for additable revolute joints in the Lego world: one is by inserting the long-pin feature found on an axle joint into a pin-hole found in bricks with holes, full beacons and gears, and the other is by inserting a short pin found in L-beacons into a partial-wide-hole feature found in flat Lego bricks with holes and in half beacons. A complete list of features found in Lego pieces that participate in kinematic connections is found in the appendix.

These features can be justified by a first-principles analysis of the configuration spaces of the interacting pieces. Such an identification process has been attempted by Faltings [Faltings90] in his paper on qualitative kinematics using configuration spaces. The configuration space construction identifies equivalence classes of positions of the two meshed gears when the teeth are in contact; free space occurs when they are not in contact. The features found in the Lego pieces could be formally justified by the construction of the configuration space of interactions between components considered pairwise.

4.1 The Refinement Method

Using feature-oriented refinement allows us to formulate a set of pure geometric constraints into a discrete constraint satisfaction problem. Thus instead of solving for contact points geometrically, we identify the sets of features needed to realize a joint in the line graph and use a local propagation method (Waltz filtering) to find a consistent set of components that implement the line graph.

Consider the problem of instantiating the line graph of `um1`. The revolute joints can be realized by using the long-pin feature and the pin-hole feature. The gear connection can be realized by two meshing gears. So the component that implements edge 1 in the line graph is a Lego brick with holes (which has the pin-hole feature) which connects to the gear using an axle-joint piece (which has the long-pin feature). The axle-joint piece fits into the axle-hole feature in a gear, realizing the revolute joint. The other revolute joint is formed in similar fashion. Thus a simple non-deterministic refinement procedure that maps edges in the line graph directly to individual components requires the following information. These tables have been constructed for the Lego world.

1. The list of features and the kinematic connections formed by feature-feature interactions.

[4] We have an animation available on videotape on request

2. The table of (feature, list of pieces where they are found) pairs.
3. The table of (piece, list of features it contains) pairs.

4.2 The Need for a More Sophisticated Refinement Procedure

The simple refinement procedure outlined in the previous subsection works best when joints are realized by the interaction of two components. Unfortunately, this is true in only the simplest of mechanisms. A joint may have some passive ("useless") degrees of freedom. For example, if we connect an axle-joint to two bricks with holes, we would theoretically have two revolute joints. However, the two bricks with holes are effectively connected by a single revolute joint. The extra rotational degree of freedom of the joint is irrelevant.

We are now in the process of developing a more sophisticated procedure that considers the use of multiple components to generate a single edge in the line graph. The issues involved in this refinement procedure are discussed briefly below.

Instead of classifying kinematic connections by the feature pairs that implement them, we construct compound feature interactions (in principle, feature interaction plans) that identify the features that participate as well as the geometric constraints satisfied by them. An instance of such a constraint is that the two features be coincident. It could also constrain the orientations of the features in space. Augmenting the feature interaction pairs in this fashion allows for generating refinements when there isn't a simple mapping between pieces and the edges in the line graph. The dimensions of the chosen links are determined used standard parametric design tools (ADAMS). The metaphorical assembly plan generated in the previous section can be used to assemble the chosen components in a feasible configuration.

5 Conclusions and Future Work

This paper has presented some of our results from a prototype program that synthesizes mechanisms from functional specifications. The key innovation was the introduction of an abstraction of kinematic function based on number of degrees of freedom, called unity machines. Unity machines allow us to apply many of the conceptual tools and techniques from functional program optimization to the problem of designing effective machines. The design of the new qualitative motion vocabulary allows for a systematic enumeration of elementary unity machines which forms the basis of our complete synthesis procedure. The synthesis procedure relies on a motion unifier which generates a variety of constraints, including geometric constraints. These are satisfied by the qualitative kinematic simulation method called degrees of freedom analysis. The extension of this analysis to fixed axis gears is presented in the Appendix. The symbolic solution to the positions of the links in a mechanism allowed for clean integration with a quantitative simulator. The issues involved in generating concrete designs were discussed, and a preliminary algorithm for producing simple refinements was given.

Unity machines and discretized finite-state unity machines account for over half of the mechanisms in Artebelovsky's catalog. This indicates the wide applicability of our abstraction to the problem of automating conceptual design in the domain of kinematics. Our future plans include the incorporation of subassembly learning, as well as the generation of design rationale to aid in the automatic modification of machine-generated designs. We will also add dynamic considerations into the design. The kinematic designs that we have considered so far have treated bodies as rigid. A new idea is to exploit elastic design to achieve flexibilities that have useful function. An example is the design of modern suspension bridges where there is flexibility along the direction of the bridge to bend until the loads in the cables on either side are equal. To incorporate these possibilities into the design space we need to include description of elastic implementation media as well as the kinematic functions that they can accomplish. This requires enriching our qualitative vocabulary of motions to include higher level descriptions of kinematic function. This is an avenue we will explore in future work, since many innovative designs involve exploiting flexible media.

Acknowledgements

CSG models of the Lego pieces were constructed using the PADL_2 [Hartquist and Marisa] language and evaluated using the RayCasting Engine (RCE) [Ellis 1991]. Rendering, mass property calculation, and null-object detection were accomplished using the Ray casting Engine [Marisa] provided by Cornell Programmable Automation, Cornell University.

References

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

[CutTan] M.R. Cutkosky and J.M. Tanenbaum, Research in Computational Design at Stanford, *Research in Engineering Design*, Vol 2, pp 53-59, Springer-Verlag, 1990.

[Ellis91] J. L. Ellis, G. Kedem, T. C. Lyerly, R. J. Marisa, J. P. Menon, D. G. Thielman, and H. B. Voelcker, *The RayCasting Engine and ray representations*, , in Proc. ACM SIGGRAPH Symposium on Solid Modeling Foundations and CAD/CAM applications, pp. 255-267 edited by J. R. Rossignac and J. Turner, June 1991.

[Faltings90] Boi Faltings. Qualitative Kinematics in Mechanisms. *Artificial Intelligence* **44** (1990), pp. 89-119.

[Freudenstein74] F. Freudenstein and L.S. Woo, Kinematic Structure of Mechanisms, in *Basic Questions of Design Theory* edited by W.R. Spillers, North Holland, 1974.

[Hartquist and Marisa] E. E. Hartquist and H. A. Marisa, *PADL_2 Users' Manual*, Cornell Programmable Automation, Ithaca, 1988.

[Hoeltzel and Chieng] D. A. Hoeltzel and W.-H. Chieng. Pattern Matching as an Automated Approach to Mechanism Design in *Theory of Machines and Mechanisms*, 1990.

[Joskowicz and Sacks] Leo Joskowicz and Elisha Sacks. Computational Kinematics. IBM Research Report RC 16869 (#74799) 5/16/91.

[Karnopp] Synthetic Dynamics: Bond Graphs in Design, in *Basic Questions of Design Theory* edited by W.R.Spillers, North Holland, 1974.

[Kramer] Glenn Andrew Kramer. Geometric Reasoning in the Kinematic Analysis of Mechanisms. TR-91-02, Schlumberger Laboratory for Computer Science, 1990; Solving Geometric Constraint Systems. In *Proceedings of the National Conference on Artificial Intelligence*, pp. 708-714, Boston, July 1990.

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

[Marisa] Richard Marisa, The RayCasting Engine User's Guide, Cornell Programmable Automation, Ithaca, 1991.

[Nilsson] N.J. Nilsson, R.E. Fikes, R. Duda, P. Hart, The STRIPS planner, in *Readings in Artificial Intelligence* edited by Bonnie Webber and N.J. Nilsson, Tioga Press, 1982.

[Orlandea] N. Orlandea, M.A. Chase, D.A Calahan, A sparsity oriented approach to the dynamic analysis and design of mechanical systems, *Journal of Engineering for Industry, Transactions ASME Ser B*, Vol 99, pp 773-784, 1977.

[Owens91] A. Owens, Technology Development: A case study of blade cleaners, Xerox Webster Design Center Internal Memo, February 1991.

[Reuleaux] M. M. Reuleaux, *The Kinematics of Machinery.* Macmillan & Co., New York, 1876. Translated by Alex B. W. Kennedy.

[Shah] J. J. Shah, Conceptual Development of Form Features and Feature Modelers, *Research in Engineering Design*, Vol 2, pp 93-108, Springer-Verlag, 1991.

[Shigley] J. E. Shigley, *Kinematic Analysis of Mechanisms*, 2nd edition, McGraw Hill, 1969.

[SriTun] S. Srikanth and J.U. Turner, Toward a Unified Representation of Mechanical Assemblies, *Engineering with Computers*, Vol 6, pp 103-112, Springer-Verlag, 1990.

[Ulrich] K. Ulrich, Computation and Pre-Parametric Design, PhD thesis, MIT AI Lab Memo 1043, July 1988.

[UlrSee] K. Ulrich and W. Seering, Conceptual Design: Synthesis of Systems of Components in *Intelligent and Integrated Manufacturing Analysis and Synthesis* edited by C.R. Liu, A. Requicha and S. Chandrasekar, ASME PED-Vol 25, 1988.

[Yang] B. Yang, U. Datta, P.Datseris, Y. Wu, An Integrated System for Design of Mechanisms by an Expert System: DOMES, *AI EDAM*, Vol 3, No 1, pp 53-70, Academic Press, 1989.

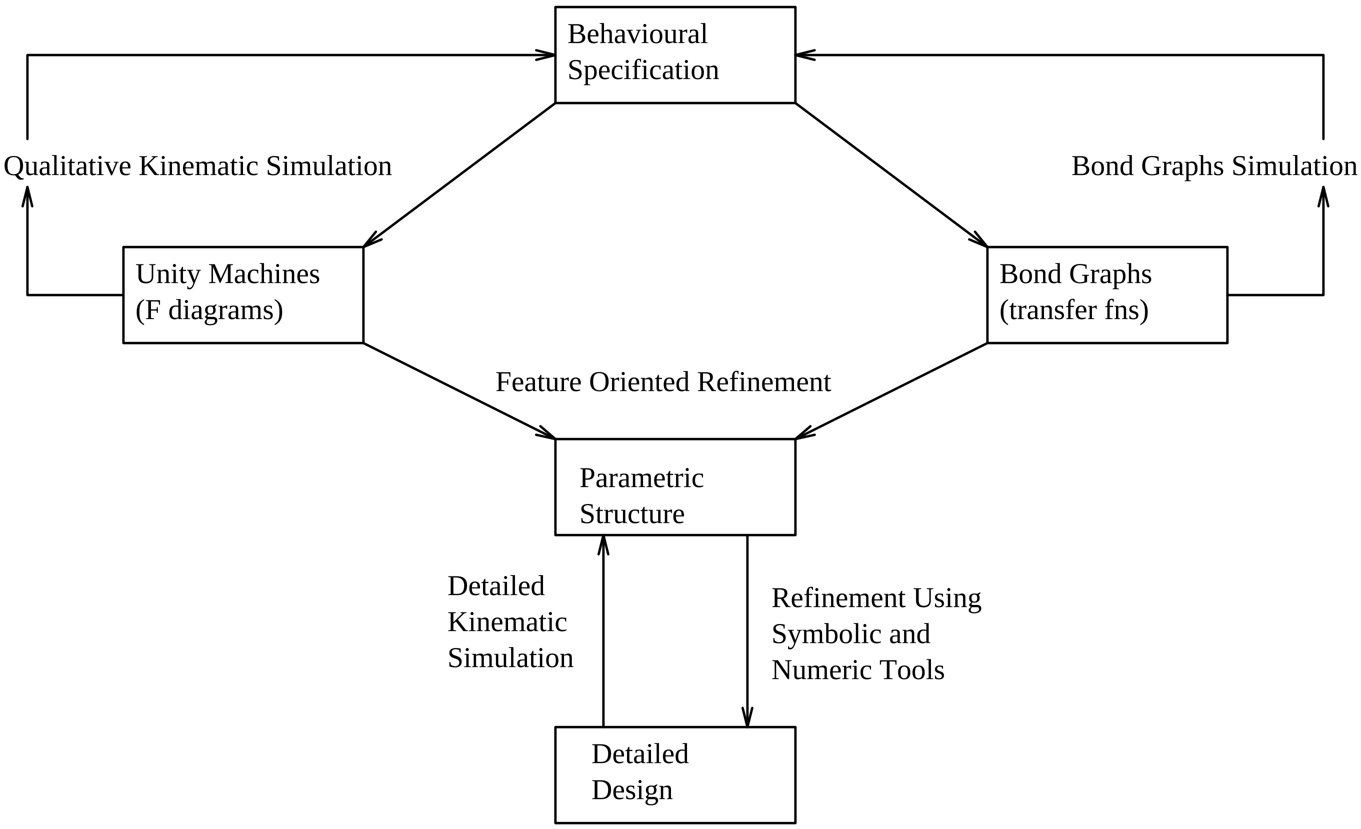


Figure 4: The Design Method

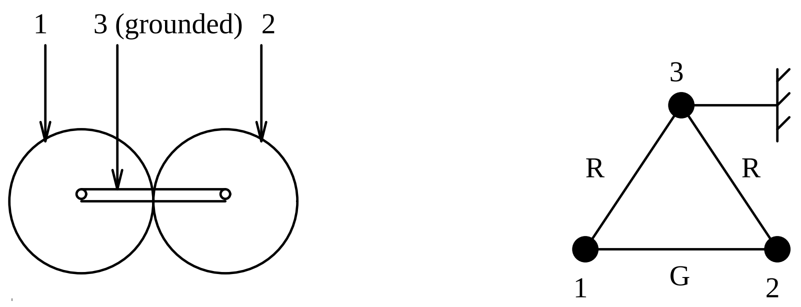


Figure 5: A simple mechanism and its F diagram

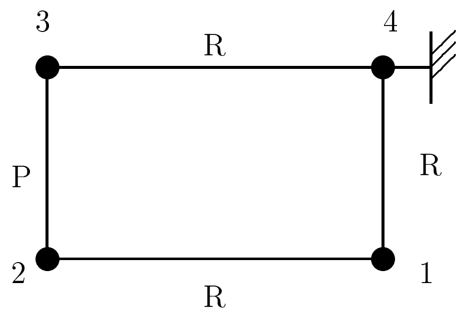


Figure 6: The F-diagram for an inversion of the slider crank mechanism

Input Motion	Output Motion	Constraints	Unity Machine

`type(M,rotation)`	`type(N,rotation)`	`sense(M,C)`
		`sense(N,not C)`
		`speed(M,S)`
		`speed(N,r` $\cdot$ `S)`
		`r` > 0
			Gear Pair(2289)
			Pin Wheel (2577)

`type(M,rotation)` `continuous(M)`	`type(N,rotation)` `intermittent(M)`	 `inplane(M,N)`	Geneva wheel (2623, 261) Pin Wheel Sprocket(2605)
`type(M,rotation)`	`type(N,rotation)` `reverse-dir(N)`	`inplane(M,N)`	4 bar linkage (539,2443) lever gear (2412)
`type(M,rotation)` `intermittent(M)`	`type(N,rotation)` `continuous(M)`		Escapement (2660) 6-bar linkage (593)
`type (M,rotation)`	`type(N,rotation)`		universal joint(554) bevel(799)

`type(M,rotation)`
`continuous(M)`
	`type(N,translation)`
	`continuous(N)`
	`reverse-dir(N)`
		`period(N,P)`
		`tpath(N,line)`
			Slider Crank(2410,2417)

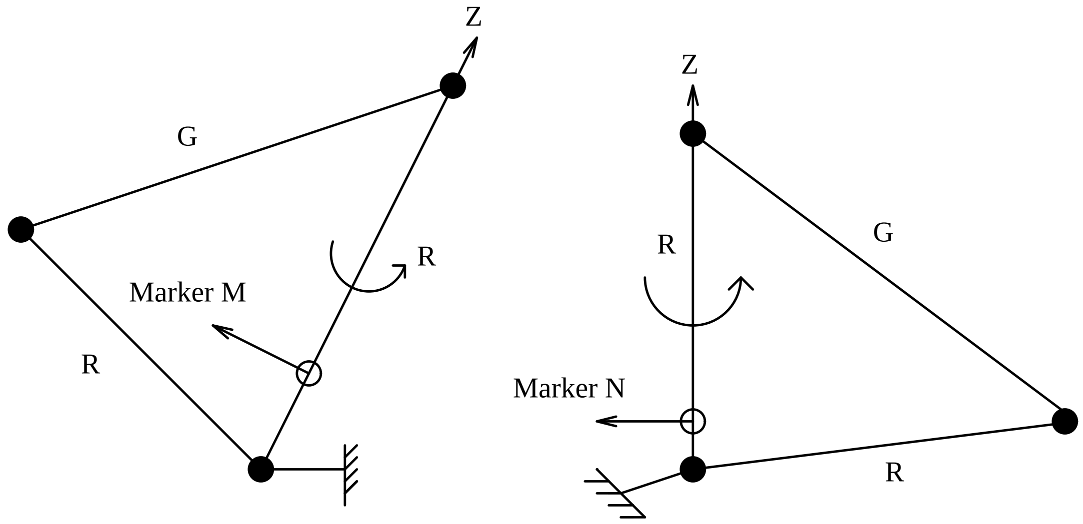
`type(M,rotation)`	`type(N,translation)` `intermittent(N)`		Intermittent Rack (2354)
`type(M,rotation)`	`type(N,translation)`	`tpath(N,line)`	Gear Teeth(2321) 4-bar mech. (644)

`type(M,rotation)`	`type(N,translation)`	`tpath(N,circle)`
		`sense(M,S)`
		`sense(N,S)`
			4-bar mech.(679)

`type(M,rotation)`	`type(N,translation)`	`tpath(N,circle)`
		`sense(M,S)`
		`sense(N,not S)`
			4-bar mech.(685)

`type(M,rotation)`	`type(N,translation)`	`tpath(N,math-fn)`	Lever gear mech.(2486)

Figure 7: Some of the Elementary Unity Machines



The Z axes of the two markers must be in line.
A marker is associated with a joint (edge) in an F-diagram.

Figure 8: Unification of two continuous rotations

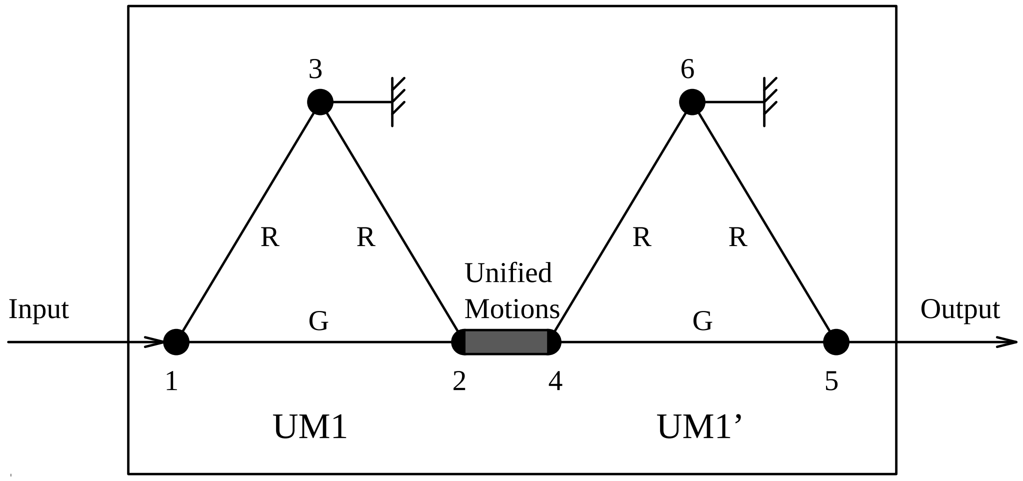


Figure 9: Composing two simple unity machines

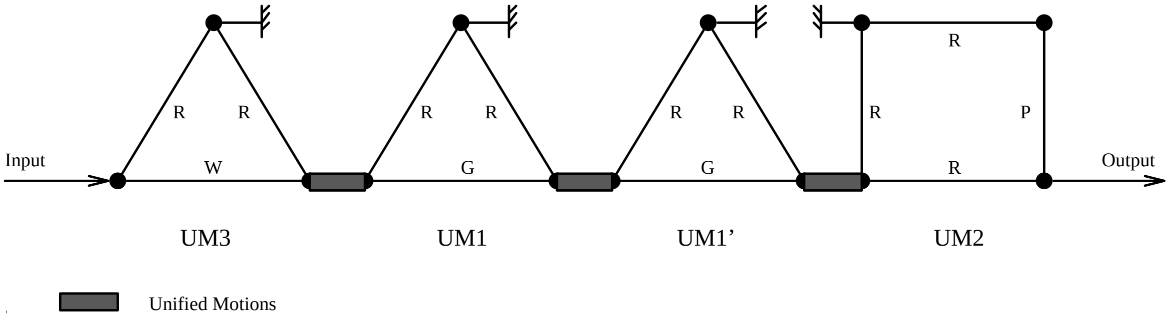


Figure 10: Synthesizing the motion of a leg of the walker

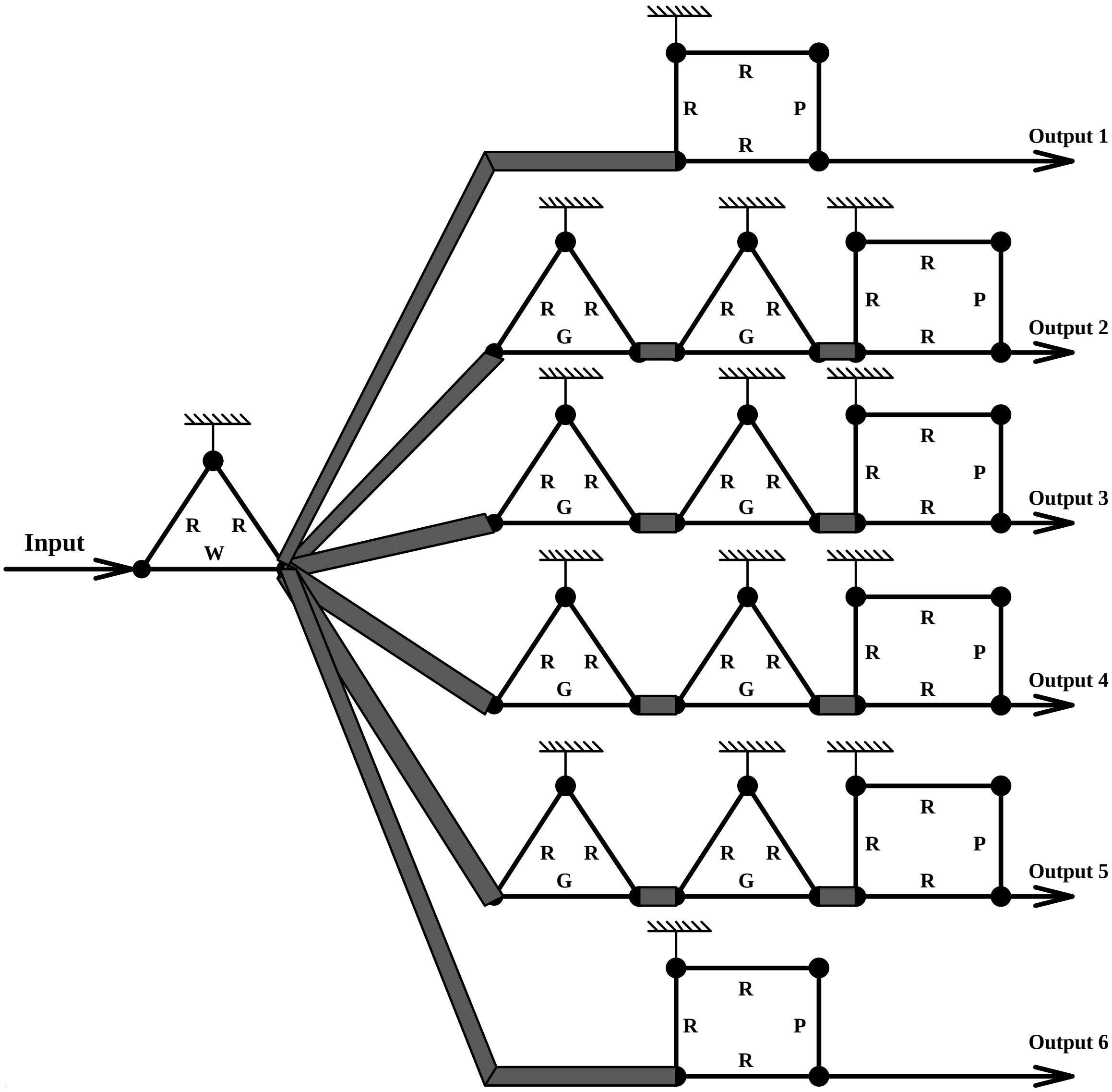


Figure 11: The optimised design of the walker

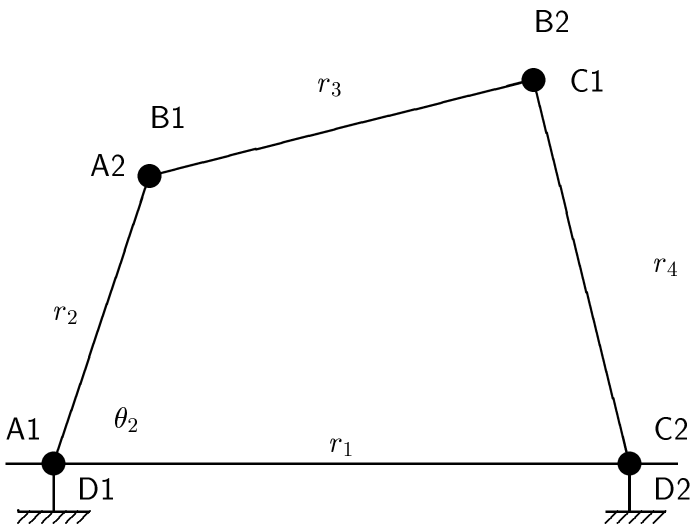


Figure 12: A four bar linkage

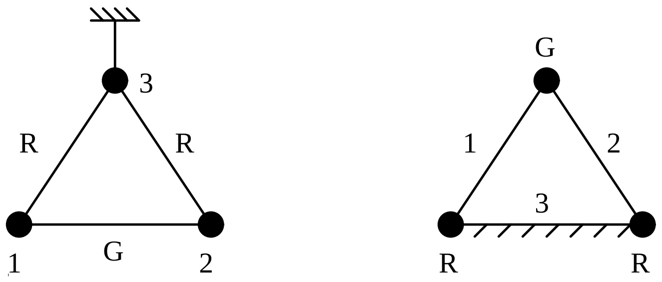


Figure 13: `um1` and its line graph

Appendix : DOF Analysis of Fixed-Axis Gears

This section describes a method of extending the kinematic analyzer to enable it to analyze mechanisms containing fixed-axis gears; that is, mechanisms in which the direction of the axis about which each gear rotates is fixed in the global coordinate frame, and in which the locations of the centers of the gears can be determined independently of the rotation of the gears. For example, these conditions are satisfied if each gear is attached by a revolute joint to a frame. Kinematic analysis is performed after the mechanism has been completely (though perhaps tentatively) designed. Thus, the input to the kinematic analyzer is a description of the mechanism in some initial configuration.

Consider two meshed gears, whose centers are represented by markers M_1 and M_2, respectively. The markers are oriented so that their z-axes are orthogonal to the plane containing the gears. The connection between these gears is described by the constraint $\texttt{meshed-rotation}(M_1, M_2, r, \alpha_1, \alpha_2)$,[5] where $r = r_1/r_2$ is the ratio of the circumferences of the gears, and α_i is the value of θ_i (defined below) in the initial configuration. This constraint is equivalent to the requirement $r_1(\theta_1 - \alpha_1) = r_2(\theta_2 - \alpha_2)$, where

$$d = \texttt{vector-difference}(\texttt{gmp}(M_2), \texttt{gmp}(M_1)) \quad \text{and} \quad \theta_i = \texttt{vector-angle}(d, \texttt{gmx}(M_i), \texttt{gmz}(M_1)).$$

The function $\texttt{vector-angle}(v_1, v_2, v_3)$ returns the angle between vectors v_1 and v_2, measured counterclockwise from v_1 to v_2 as viewed from the positive direction of v_3. Thus, this `meshed-rotation` constraint is expanded into

$\texttt{aux-marker}(X, \texttt{vector-sum}(\texttt{gmp}(M_1), \texttt{vector-scale}(d, 0.5)), \texttt{gmz}(M_1), \texttt{unit-vec}(d))$
$\texttt{offset-x}(X, M_2, r \cdot (\theta_1 - \alpha_1) + \alpha_2)$
$\texttt{offset-x}(X, M_1, r \cdot (\theta_2 - \alpha_2) + \alpha_1)$

Here, d and θ_i are just abbreviations for the expressions given above. `aux-marker` defines a new marker whose location, z-axis, and x-axis are determined by evaluating the given expressions. X represents some currently unused name.

The constraint $\texttt{meshed-driving-input}(M_1, M_2, s)$ is used to specify the values of driving inputs for these connections. This constraint is equivalent to the requirement $\theta_1 = s$, where θ_i is as defined above. The value of s is one of the driving inputs of the mechanism.

The DOF analysis proceeds as follows.

1. All of the constraints not involving meshed rotations are satisfied.

2. The `meshed-rotation` constraints are expanded as described above, except the `offset-x` constraint is omitted for markers for which there is a `meshed-driving-input` constraint in the problem database.

3. Each constraint $\texttt{meshed-driving-input}(M_i, M_j, s_{ij})$ is satisfied by rotating the link containing marker M_i.

4. The remaining `offset-x` predicates are satisfied in the usual DOF-analysis manner, which is:

[5] Note that we do not include here the predicates that determine the locations or the orientations of the z-axes of the markers, since (as mentioned above) we assume that these degrees of freedom are removed by other constraints in the problem database, that these constraints can be satisfied by the DOF analyzer independently of the remaining rotational DOF of each gear, and that this has been done already. When it starts processing the `meshed-rotation` constraint, the analyzer can easily verify that the gears are positioned and oriented so that their teeth mesh correctly.

(a) Choose an unsatisfied constraint.

(b) Check whether the preconditions (assumptions) of the corresponding PFT entry are satisfied.

(c) If so, satisfy the constraint using the plan fragment; otherwise, choose a different constraint.

The `offset-x` constraint between two markers can be satisfied only when the orientation of one of them is known. Thus, as the analysis proceeds, phase information will get propagated from links whose phase was determined by a driving input along `meshed-rotation` connections to the remainder of the mechanism. Note that if too few driving inputs were specified, there will be unconstrained degrees of freedom remaining when the analysis terminates. If too many were specified, an inconsistency (due to overconstraint) may occur while determining the orientation of a link; this situation will be detected and reported as an error.