Artificial Intelligence 104 (1998) 211–264

Artificial Intelligence

Generating multiple new designs from a sketch

Thomas F. Stahovich [a,1,*], Randall Davis [b], Howard Shrobe [b]

[a] *Department of Mechanical Engineering, Carnegie Mellon University, Pittsburgh, PA 15213, USA*
[b] *MIT Artificial Intelligence Laboratory, 545 Technology Square, Cambridge, MA 02139, USA*

Received 13 August 1997; received in revised form 21 May 1998

Abstract

We describe a program called SKETCHIT that transforms a single sketch of a mechanical device into multiple families of new designs. It represents each of these families with a “BEP-Model”, a parametric model augmented with constraints that ensure the device produces the desired behavior. The program is based on qualitative configuration space (qc-space), a novel representation that captures mechanical behavior while abstracting away its implementation. The program employs a paradigm of abstraction and resynthesis: it abstracts the initial sketch into qc-space, then uses a library of primitive mechanical interactions to map from qc-space to new implementations. © 1998 Elsevier Science B.V. All rights reserved.

Keywords: Sketch understanding; Design generalization; Mechanical design; Qualitative geometric reasoning

1. Introduction

SKETCHIT is a computer program capable of taking a single sketch of a mechanical device and generalizing it to produce multiple new designs. The program’s input is a stylized sketch of the design and a description of the desired behavior; from this it generates multiple families of new designs.

The program does this by first transforming the sketch into a representation that captures the behavior of the original design while abstracting away its specific implementation. The program then uses a library of primitive mechanical interactions to map from this abstract representation to multiple new families of implementations. This representation, which

* Corresponding author. Email: stahov@andrew.cmu.edu.

[1] Support for this project was provided by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract N00014-91-J-4038.

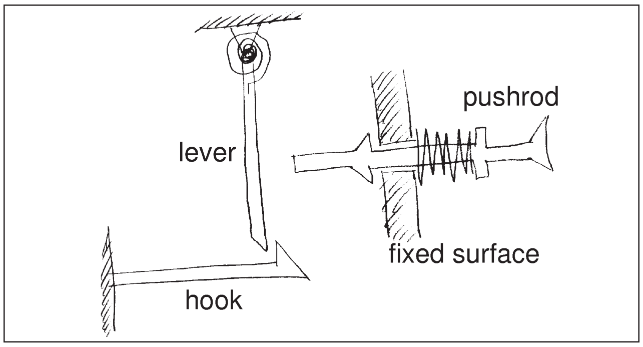


Fig. 1. A pencil sketch of a circuit breaker.

we call qualitative configuration space, is the key tool allowing SKETCHIT to perform its tasks.

The program represents each of the new families of implementations with what we call a behavior ensuring parametric model ("BEP-Model"): a parametric model augmented with constraints that ensure the geometry produces the desired behavior.[2] Our program thus takes as input a single sketch of a device and produces as output multiple BEP-Models, each of which will produce the desired behavior.

As we illustrate below, SKETCHIT's ability to generalize a single sketch into multiple families of new designs is useful for a variety of reasons. During conceptual design, for example, a high premium is placed on examining a large number of alternatives. SKETCHIT aids in this case by automatically generating a large variety of designs. Later in the design process SKETCHIT can assist the designer in adapting an initial design to meet other design requirements such as those on size or performance.

The next section uses the design of a circuit breaker to illustrate the input to and output from the program. Later, Section 9 uses the design of a dwell mechanism (yoke and rotor device) to illustrate how the program assists the designer in refining an initial design to meet specific performance requirements.

1.1. Example: circuit breaker

In this section we show SKETCHIT in action designing a circuit breaker. We begin with the program input: a stylized sketch of the device and a state transition diagram describing the desired behavior. We conclude this section with three of the new designs that SKETCHIT produces when it finally maps the circuit breaker's qc-space back to geometry.

Fig. 1 shows a pencil sketch of one implementation for a circuit breaker. In normal use, current flows from the lever to the hook; current overload causes the bimetallic hook to heat and bend, releasing the lever and interrupting the current flow. After the hook cools, pressing and releasing the pushrod resets the device.

[2] A parametric model is a geometric model in which the shapes are controlled by a set of parameters.

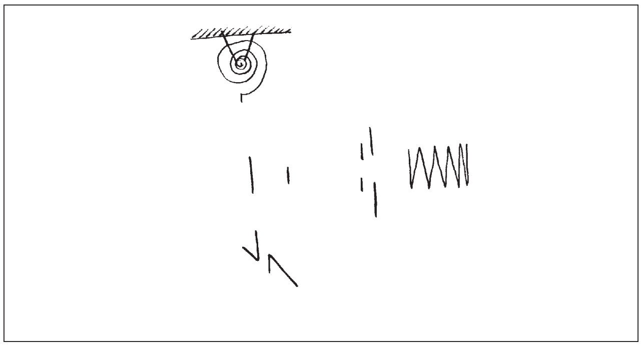


Fig. 2. Stripping away the non-functional parts of Fig. 1.

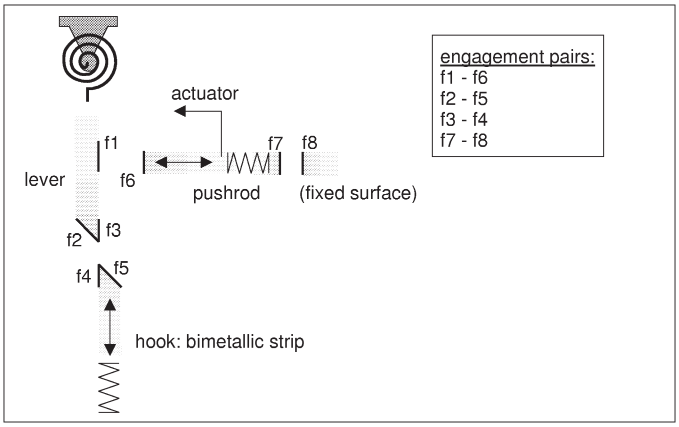


Fig. 3. Stylized sketch of the circuit breaker as actually input to program. Engagement faces are bold lines. The actuator applied to the pushrod represents the reset motion imparted by the user. For our convenience we use labels to refer to engagement pairs: (f1 f6) = push-pair, (f2 f5) = cam-follower, (f3 f4) = lever-stop, (f7 f8) = pushrod-stop.

SKETCHIT is concerned with only the functional parts of the sketch: springs, actuators, kinematic joints, and the faces where parts meet and through which force and motion are transmitted. Fig. 2 shows what is left when we peel away all but the functional parts of the pencil sketch. This is the sort of information that is contained in the stylized sketch that SKETCHIT takes as input.

Our software currently provides a mouse driven sketching interface for creating stylized sketches.[3] Fig. 3 shows the stylized version of the circuit breaker that the designer

[3] Eventually we will develop a pen based sketching interface allowing the designer to sketch directly on a digitizing pad with a stylus. The designer will draw the usual sort of sketch (i.e., like the one in Fig. 1) and then annotate it with the stylus to indicate the functional parts.

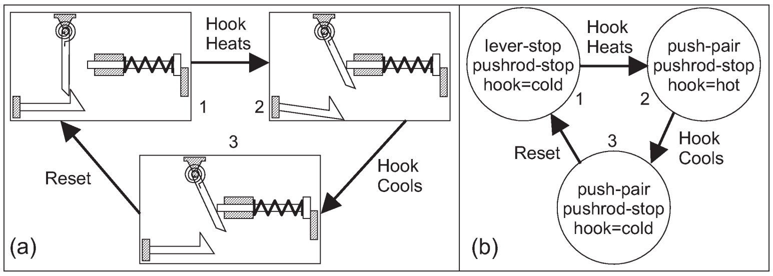


Fig. 4. The desired behavior of the circuit breaker. (a) Physical interpretation. (b) State transition diagram. In each of the three states, the hook is either at its hot or cold neutral position.

creates with this interface.[4] Line segments are used for part faces; icons are used for springs, joints, and actuators. SKETCHIT focuses on just the part-faces that are functional; consideration of the connective geometry (the surfaces that connect the functional faces to make complete solids) is put off until later in the design process. The designer annotates the stylized sketch to indicate which pairs of faces are intended to engage each other (the annotations are listed in the table contained in the figure).

The designer describes the desired behavior of a device to SKETCHIT using a state transition diagram. Each node in the diagram is a list of the pairs of faces that are engaged and the springs that are relaxed.[5] The arcs are the external inputs that drive the device. Fig. 4(b), for instance, describes how the circuit breaker should behave in the face of heating and cooling the hook and pressing the reset pushrod.

Fig. 5 shows a portion of one of the BEP-models that SKETCHIT derives from the sketch of the circuit breaker and the desired behavior. The top of the figure shows the parameters that define the sloped face on the lever (f2) and the sloped face on the hook (f5). The bottom shows the constraints that ensure this pair of faces plays its role in achieving the overall desired behavior: i.e., moving the lever clockwise pushes the hook down until the lever moves past the point of the hook, whereupon the hook springs back to its rest position. As one example of how the constraints enforce the desired behavior, the ninth equation,

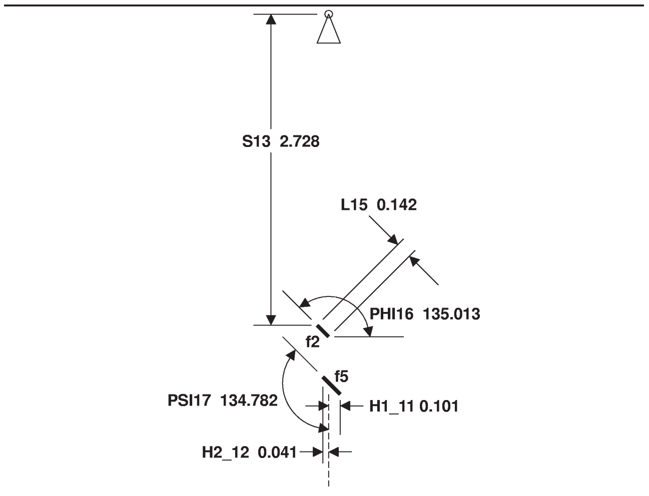
```
0 > R14/TAN(PSI17) + H2_12/SIN(PSI17),
```

constrains the geometry so that the contact point on face f2 never moves tangent to face f5. This in turn ensures that when the two faces are engaged, clockwise rotation of the lever always increases the deflection of the hook.

The parameter values shown in the top of Fig. 5 are solutions to the constraints of the BEP-Model, hence this particular geometry provides the desired behavior. The values were computed by a program called DesignView, a commercial parametric modeler based on variational geometry. (For our purposes, DesignView is simply a non-linear equation solver which is also capable of drawing lines and arcs.) Using DesignView to interactively

[4] In the remainder of this document, we use the term sketch to refer to the kind of stylized sketch shown in Fig. 3.

[5] The pairs of faces not listed at a node are by default disengaged, the springs not listed are by default not relaxed.



```
H1_11 > 0      H2_12 > 0      S13 > H1_11
L15 > 0        PHI16 > 90     PHI16 < 180
PSI17 > 90     PSI17 < 180
0 > R14/TAN(PSI17) + H2_12/SIN(PSI17)
R14 = SQRT(S13^2 + L15^2 - 2*S13*L15*COS(PHI16))
```

Fig. 5. Output from the program (a BEP-Model). Top: the parametric geometry; the decimal number next to each parameter is the current value of that parameter. Bottom: the constraints on the parameters. For clarity, only the parameters and constraints for faces f2 and f5 are shown.

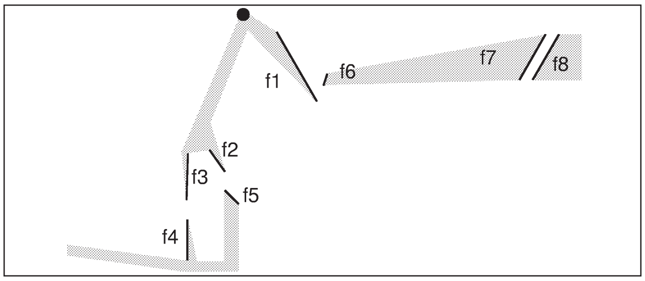


Fig. 6. Another solution to the BEP-Model of Fig. 5. Shading indicates how the faces might be connected to flesh out the components. This solution shows that neither the pair of faces at the end of the lever (f2 and f3) nor the pair of faces at the end of the hook (f4 and f5) need be contiguous.

adjust parameter values we can easily explore the family of designs defined by the BEP-Model. Fig. 6, for example, shows another solution to this BEP-Model. Because these parameter values satisfy the BEP-Model, even this rather unusual geometry provides the desired behavior. As this example illustrates, the family of designs defined by a BEP-Model includes a wide range of design solutions, many of which would not be obtained with conventional approaches.

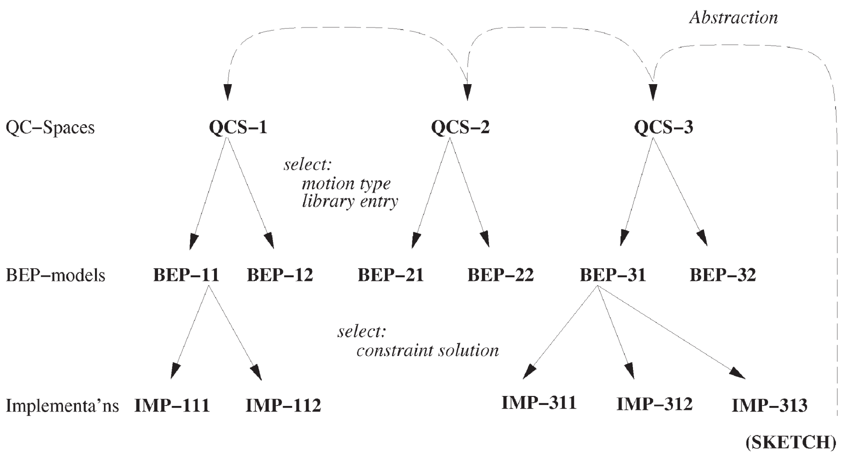


Fig. 7. Overview of SKETCHIT's abstraction and instantiation process. The initial sketch is abstracted into one or more qc-spaces, each of which can produce one or more BEP-models, each of which may have one or more solutions to its set of constraints.

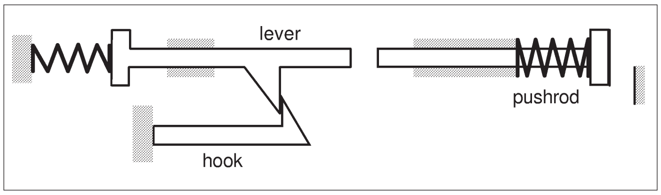


Fig. 8. A design variant obtained by replacing the rotating lever with a translating part.

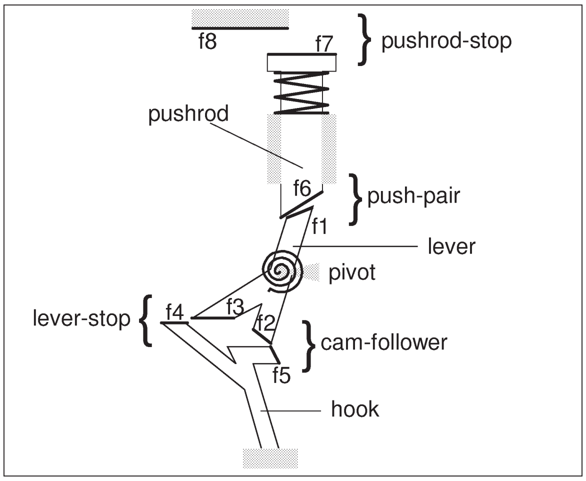


Fig. 9. A design variant obtained by using different implementations for the engagement faces. In the position shown, the pushrod is pressed so that the hook is just on the verge of latching the lever.

Figs. 5 and 6 show members of just one of the families of designs that the program produces for the circuit breaker. SKETCHIT produces other families of designs (i.e., other BEP-Models) by selecting different motion types (rotation or translation) for the components and by selecting different implementations for the pairs of interacting faces (Fig. 7). For example, Fig. 8 shows a design obtained by selecting a new motion type for the lever: in the original design the lever rotates, in the new design it translates. Fig. 9 shows an example of selecting different implementations for the pairs of interacting faces: In the original implementation of the cam-follower engagement pair, the motion of face f2 is roughly perpendicular to the motion of face f5; in the new design of Fig. 9, the motions are parallel. Conversely, the motions of faces f1 and f6, originally parallel, are now perpendicular.

2. Representation: qc-space

SKETCHIT's task is to generalize a single design into multiple new designs. The new designs should provide the same behavior as the original but employ new implementations. Our approach to this task was to develop a representation that captures the behavior of the original design while abstracting away its particular implementation, providing the opportunity to select new implementations.

For the class of devices that SKETCHIT is concerned with, rigid body devices with springs and negligible inertia, the overall behavior is determined by the interactions between the components. It is not possible to determine the behavior of a single component without knowing the other components with which it interacts. For example, the component in Fig. 10 can exhibit a wide range of behaviors depending on what it interacts with: if it interacts with a broom handle and a stop as in Fig. 11(a) its behavior is self energizing friction which prevents slipping and holds the handle in place. If it interacts with a translating flat face as in Fig. 11(b) it provides cam and follower behavior.

For many types of devices, such as classical electrical circuits, it is possible to exhaustively enumerate the possible kinds of behavior a given component can exhibit. For example, transistors are typically idealized as having just a few distinct operating modes. However, the kinds of devices we are considering do not afford this same compactness of description. A given mechanical component can exhibit a very large number of possible behaviors because there is a near infinite variety of other components with which it could interact. (Fig. 11 shows just two of possible kinds of behavior of the curved, pivoted object.) Thus, instead of attempting to represent behavior at the level of individual components, we instead focus directly on the interactions between components.

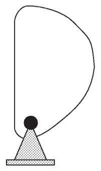


Fig. 10. A curved object with a pivot.

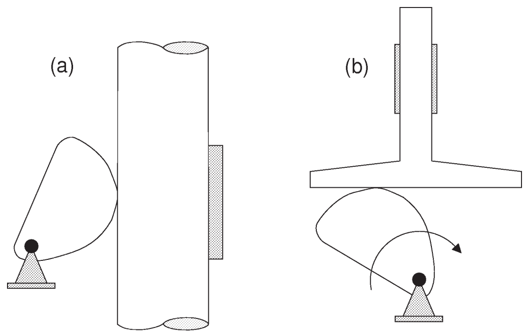


Fig. 11. (a) A broom handle holder. (b) A cam and translating flat-face follower.

Our search for a representation began with configuration space (c-space), which is commonly used to represent this kind of behavior. But, while c-space is capable of representing the behaviors we are interested in, it does not adequately abstract away their implementations, and thus does not provide adequate opportunity for selecting new implementations. We discovered that abstracting c-space into a qualitative form produces the desired effect; hence we call SKETCHIT's behavioral representation "qualitative configuration space" (qc-space).

This section begins with a description of c-space, then describes how we abstract c-space to produce qc-space.

2.1. C-space

Consider the rotor and slider in Fig. 12. If the angle of the rotor U_R and the position of the slider U_S are as shown, the faces on the two bodies will touch. These particular values of U_R and U_S are termed a *configuration* of the bodies in which the faces touch, and can be represented as a point in the plane, called a configuration space plane (cs-plane), shown in Fig. 13.

If we determine all of the configurations of the bodies in which the faces touch and plot the corresponding points in the cs-plane (Fig. 13), we get a curve, called a configuration space curve (cs-curve). The shaded region "behind" the curve indicates blocked space, configurations in which one body would penetrate the other. The unshaded region "in front" of the curve represents free space, configurations in which the faces do not touch.

In general, c-space represents the configurations of a device in which the bodies do not touch (free space), the configurations in which the bodies would interpenetrate (blocked space), and the configurations in which the bodies just touch (the boundary between free and blocked space). The axes of the c-space are the position parameters of the bodies; the dimension of the c-space is the number of degrees of freedom of the set of bodies.

To simplify geometric reasoning in c-space, we assume that devices are fixed-axis. That is, we assume that each body either translates along a fixed axis or rotates about a fixed axis. Hence in our world the c-space for a pair of bodies (i.e., a "pairwise c-space") will always be two-dimensional and the boundary between blocked and free space will always be a curve (a cs-curve).

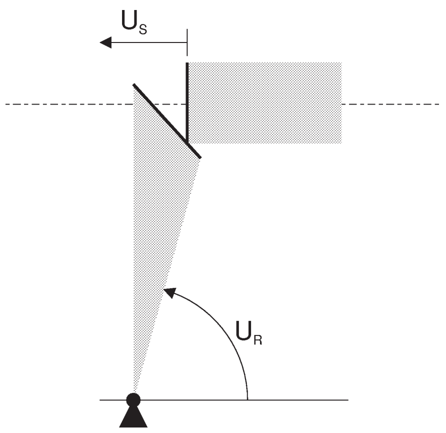


Fig. 12. A rotor and slider. The slider translates horizontally. The interacting faces are shown with bold lines.

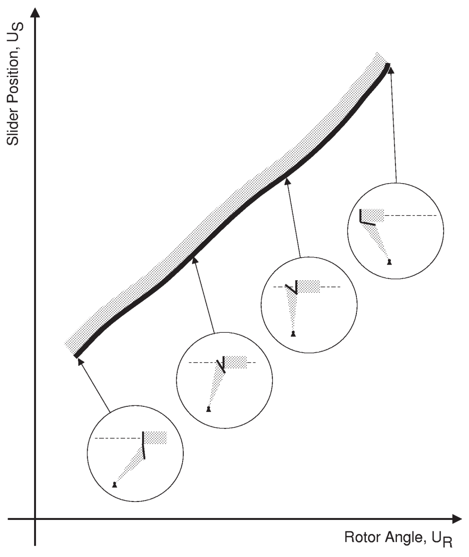


Fig. 13. The c-space of the rotor and slider. The inset figures show the configuration of the rotor and slider corresponding to selected points on the cs-curve.

Although a pairwise c-space will always be two-dimensional, its topology will be planar only if both bodies either rotate less than a full revolution or translate. If one of the bodies rotates through full revolutions, the c-space will be a cylinder; if both rotate through full revolutions it will be a torus. For example, if the rotor and slider in Fig. 12 start from some initial configuration (with the slider far from the rotor), and the rotor angle increases through a full revolution, the device will be back in its initial configuration. For this to happen, the right edge of the cs-plane must wrap around and connect to the left edge to form a cylinder, as shown in Fig. 14.

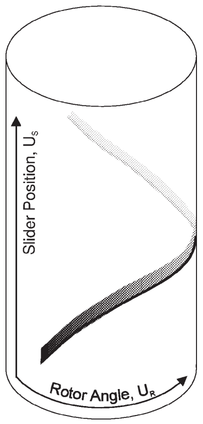


Fig. 14. If the rotor turns through a full revolution, the c-space becomes a cylinder.

SKETCHIT represents non-planar, pairwise c-spaces by flattening them out into planes and imposing periodic boundary conditions (see Section 4.3.1).[6] We use the term cs-plane to refer to all pairwise c-spaces: those that are naturally planar as well as those that are flattened out representations of non-planar c-spaces.

2.1.1. Representing springs, actuators, and fixed surfaces

The devices that SKETCHIT is intended to handle may contain springs, actuators, and interactions with fixed surfaces in addition to the moving faces described above. This section describes how c-space represents the behavior of these other kinds of components.

Consider the device in Fig. 15, consisting of two blocks, A and B, that interact through a pair of sloped engagement faces. A spring is attached to block A, an actuator (an external motion source) is attached to block B, and there is a fixed surface with which B may collide.

We start by applying the techniques of the previous section to obtain a representation of the interaction between the faces of A and B. In the configuration shown, the faces are touching, and thus this configuration is a point on the cs-curve for the engagement pair. If A moves a small distance in the positive direction, it will push B a small distance in the positive direction (assume the actuator is turned off). The faces will still be engaged in this new configuration, and hence we have another point on the cs-curve. Continuing in this fashion we obtain a diagonal cs-curve with positive slope, as the cs-plane in Fig. 15 shows. If the faces are touching and A moves in the positive direction while B is held fixed, the faces will penetrate each other. Hence, the space to the right of the cs-curve is blocked space. (Alternatively, if B moves in the negative direction while A is held fixed, the faces will penetrate. Hence, the space below the cs-curve is blocked space. Either way, we compute the same blocked space.)

Next consider the spring attached to block A. At some position of block A (call it np), the spring will be at its neutral position (i.e., the position at which the spring is relaxed).

[6] We have implemented procedures for handling cylindrical pairwise c-spaces but not toroidal ones. However, the techniques for handling toroidal c-spaces are a direct extension of those we use for cylindrical c-spaces.

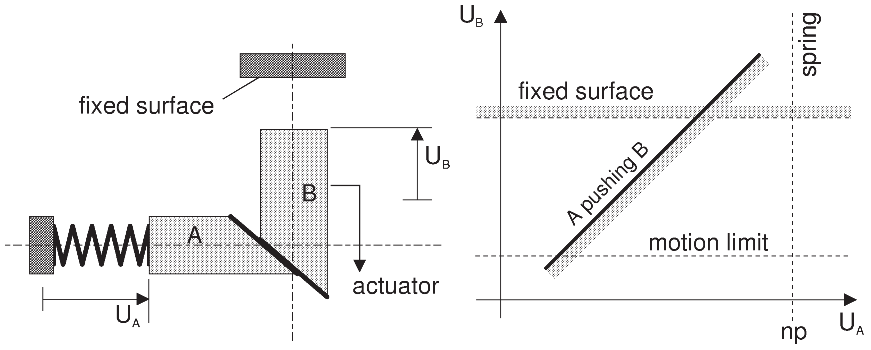


Fig. 15. A simple mechanical system and its configuration space.

Because the neutral position depends only on the position of A, it can be viewed in the cs-plane as the vertical line $U_A = np$.

The neutral position of a spring always appears as a vertical or horizontal line in the cs-plane, because in SKETCHIT's world a spring always has one end fixed, and hence the relaxed state of a spring depends on the position of only one body.

In SKETCHIT's world, actuators apply a motion to a body until the body reaches a particular position called the motion limit of the actuator. When the body reaches this position the actuator turns off, no longer applying any force or motion to the body. Because the motion limit of an actuator depends on the position of only one body, a motion limit similarly appears as a vertical or horizontal line in the cs-plane, e.g., the horizontal line in Fig. 15.

Finally, consider the interaction between the fixed surface and block B. (Because the actuator pushes B away from the fixed surface, this interaction will not happen until after the actuator turns off.) Because this interaction depends only on B's position (the fixed surface does not move), the corresponding cs-curve is an infinite horizontal line in the cs-plane. Because the fixed surface limits B's motion in the positive direction, there is blocked space above the cs-curve.

As Fig. 15 illustrates, a cs-plane may contain both finite cs-curves representing the interaction between a pair of faces (the diagonal cs-curve), and infinite boundaries representing spring neutral positions, motion limits of actuators, and interactions with fixed surfaces. As a means of differentiation, we use solid lines for finite cs-curves and dashed lines for infinite boundaries. For convenience we refer to both finite cs-curves and infinite boundaries as cs-curves.

2.1.2. Computing motion using c-space

The previous sections described how c-space represents the behavior of all the different kinds of parts that compose the devices in our world. Our next concern is how to determine the overall behavior of a device from the description of the part behaviors. For mechanical devices composed of rigid bodies and springs, the overall behavior is characterized by the time history of the motion of each rigid body in the device.

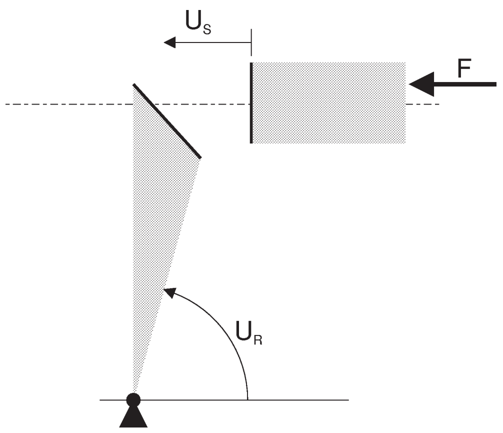


Fig. 16. Force F pushes the slider to the left.

We compute the time history of the motion by applying Newton's laws directly in c-space using what we call the particle metaphor. We illustrate the process with the device in Fig. 16 consisting of a rotor and slider. To provide a point of reference, we first compute the motion of the rotor and slider by reasoning directly from their structure. Then, we repeat the process, this time using the particle metaphor to compute the motion directly from the c-space description of the device.

Imagine that the rotor and slider are in the initial positions shown in Fig. 16 and that a force (F) pushes the slider to the left. As it moves, the slider will strike the rotor and begin to push it out of the way. Assuming the collision is inelastic, the rotor and slider will remain in contact, and the slider will continue to push the rotor out of the way. Eventually the slider will push the rotor far enough that the two parts disengage. Assuming the motion of the rotor is inertia-free, the rotor will then stop and the slider will continue to move.

We can describe the motion of the rotor and slider as a sequence of configurations in c-space. We call this sequence the *trajectory* through c-space. Fig. 17(a) shows the trajectory corresponding to the motion described above. In the initial configuration the rotor and slider are not touching, hence the initial configuration is in free space. When the motion begins, only the slider moves, and the trajectory is vertical. While the slider pushes the rotor, the configuration is a point on the cs-curve and thus the trajectory follows the curve. Once the engagement is broken, only the slider moves; the trajectory is once again vertical.

In the physical world forces cause bodies to move, the c-space view is that forces cause the configuration to change. To compute dynamics directly from c-space we treat the configuration as a particle in the cs-plane and apply forces directly to this particle. Cs-curves act like physical surfaces that deflect the particle.

Using the particle metaphor we can now compute the motion of the rotor and slider directly from their c-space, without any reference to the structure of the device. Fig. 17(b) shows the particle we use to represent the configuration of the rotor and slider. The force applied to the slider appears in the cs-plane as a force in the direction of the slider's c-space coordinate (vertical). Similarly, any forces applied to the rotor would appear in the cs-plane

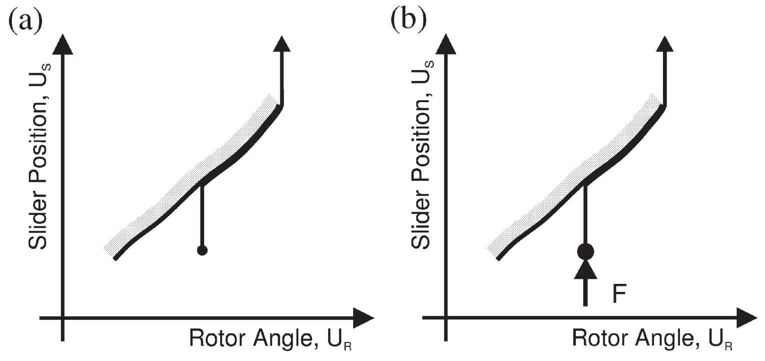


Fig. 17. (a) The motion of the rotor and slider can be represented as a trajectory through c-space. The initial configuration is shown as a dot. (b) The motion of the rotor and slider can be computed by treating the *configuration* as a particle with forces applied to it.

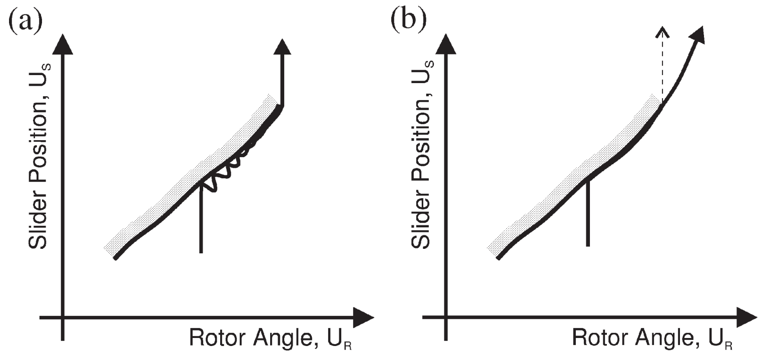


Fig. 18. (a) The trajectory of the rotor-slider device if collisions are elastic. (b) The trajectory of the rotor-slider device if there is appreciable inertia.

in the direction of the rotor's c-space coordinate (horizontal). The initial configuration of the rotor and slider defines the initial location of the particle. The net force on the particle causes it to move vertically, until it strikes the cs-curve. The cs-curve deflects the particle as it continues to move upward. Eventually, the particle reaches the end of the cs-curve and once again moves directly upward.

This example illustrates a general principle: using the particle metaphor, it is always possible to compute the motion of the bodies in a device directly from the device's c-space description.

Simplifying assumptions. As the example in Fig. 17(b) demonstrated, we make simplifying assumptions about the motion of bodies. We assume that collisions are inelastic, that motion is inertia-free, and that contacts are frictionless (we did not explicitly mention the frictionless assumption in this example). If the collision between the slider and rotor were elastic, the rotor would bounce off the slider and the trajectory in c-space might look like Fig. 18(a). If there were appreciable inertia, the rotor would continue to move after it disengaged the slider, producing a trajectory like the one in Fig. 18(b). If there were substantial friction between the rotor and slider, the slider might not be able to push the rotor and the device would jam.

We make these assumptions about the motion of bodies because they greatly simplify the trajectories through c-space and hence simplify reasoning about the trajectories. These assumptions are not overly restrictive: Sacks and Joskowicz [39] examined 2500 mechanisms in a catalog of mechanisms and found that 80% could be modeled accurately with these assumptions.

2.2. Abstracting to qc-space

C-space has many of the properties we require of a behavioral representation: it can represent the behavior of all of the components in SKETCHIT's domain, and it can be used to reason about behavior, allowing us to compute the overall behavior of a device directly from the c-space description of its parts.

To facilitate synthesizing new designs we require a behavioral representation with one other essential property: it must generalize the design by abstracting away the implementation of the behaviors.

C-space does this to a small degree. Any pair of faces that produces the cs-curve in Fig. 13 will produce the same behavior (i.e., the same dynamics) as the original pair of faces in Fig. 12. For example the slider's face can be extended upward. Thus, each cs-curve represents a family of interacting faces that all produce the same behavior.

We can, however, identify a much larger family of faces that produce the same behavior by abstracting the numerical cs-curves to obtain a qualitative c-space. In qualitative c-space (qc-space) cs-curves are represented by their qualitative slopes and the locations of the curves relative to one another. By qualitative slope we mean the obvious notion of labeling monotonic curves as diagonal (with positive or negative slope), vertical, or horizontal; by relative location we mean relative location of the curve end points. [7]

To see how even a qualitative representation of slope captures something essential about the behavior, we return to the rotor and slider. The essential behavior of this device is that the slider can push the rotor: positive displacement of the slider causes positive displacement of the rotor, and negative displacement of the rotor causes negative displacement of the slider. If the motions of the rotor and slider are to be related in this fashion, their cs-curve must be a diagonal curve with positive slope. Conversely, any geometry that maps to a diagonal curve with positive slope will produce the same behavior as the original design.

Their are eight types of qualitative cs-curves (qcs-curves) as shown in Fig. 19. Diagonal curves always correspond to pushing behavior; vertical and horizontal curves correspond to what we call "stop behavior", in which the extent of motion of one part is limited by the position of another. Each qcs-curve represents a family of monotonic cs-curves that all have the same qualitative slope.

In c-space cs-curves and infinite boundaries have absolute locations, but in qc-space locations are relative. A pair of landmark values defines the location of each end point of

[7] We restrict qcs-curves to be monotonic to facilitate qualitative simulation of a qc-space. See Section 5.

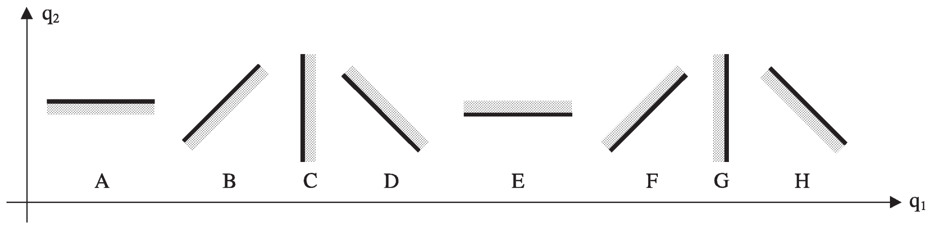


Fig. 19. There are eight types of qcs-curves. For convenience, the diagonal curves are drawn as straight lines, but they represent any diagonal, monotonic curve.

a qcs-curve; one landmark value defines the axis crossing of each infinite boundary. The ordering of the landmark values encodes the relative locations of all qcs-curves and infinite boundaries.

2.2.1. Qc-space a design space for behavior

An important consequence of generalizing each cs-curve to a family consisting entirely of *monotonic* curves is that the qualitative slopes and the relative locations completely determine the first order dynamics of the device. By first order dynamics we mean the dynamic behavior obtained when the motion is assumed to be inertia-free and the collisions are assumed to be inelastic and frictionless.[8] Thus qc-space captures *all* of the relevant physics of the overall device, and hence serves as a design space for behavior. It is a particularly convenient design space because it has only two properties: qualitative slope and relative location.

Another important feature of qc-space is that it is constructed from a very small number of building blocks, viz., the different types of qcs-curves in Fig. 19. As a consequence we can easily map from qc-space back to new implementations using precomputed implementations for each of the building blocks. We show how to do this in Section 8.

3. System

Fig. 20 illustrates the abstraction and resynthesis paradigm that SKETCHIT uses to transform a sketch into multiple new designs. In the abstraction phase SKETCHIT reverse engineers and generalizes the original design, producing a qualitative configuration space representation of the design. In the resynthesis phase SKETCHIT generates multiple implementations for the design by working from the qc-space representation it created.

SKETCHIT uses abstraction to simplify the reasoning process: a sketch contains an enormous amount of geometric detail, much of which is irrelevant to the behavior. The abstraction process strips away the irrelevant detail to expose what is essential.

[8] "Inertia-free" refers to the circumstance in which the inertia terms in the equations of motion are negligible compared to the other terms, perhaps due to high friction, low mass, or large applied forces. One important property of inertia-free motion is that there are no oscillations. This set of physical assumptions is also called quasi-statics.

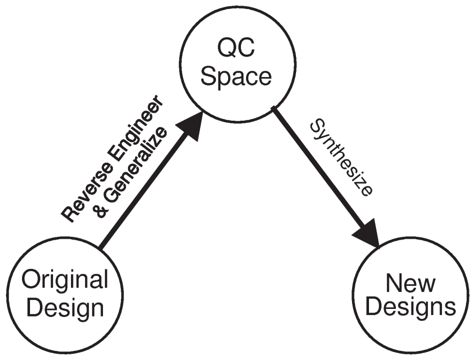


Fig. 20. The problem solving paradigm.

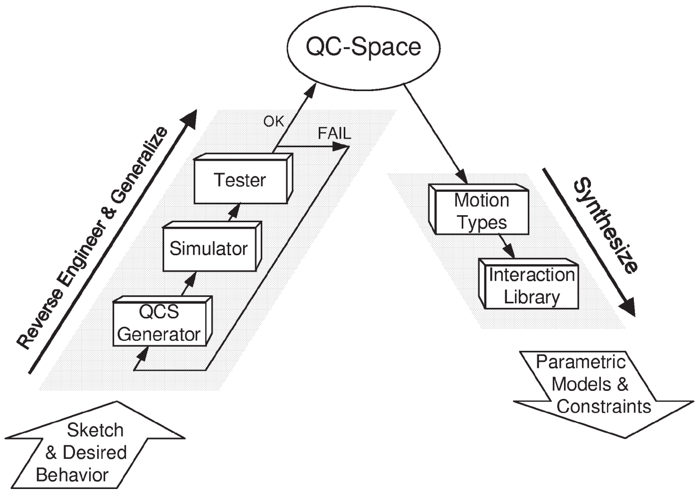

QC-Space
OK
FAIL
Tester
Simulator
QCS Generator
Reverse Engineer & Generalize
Motion Types
Interaction Library
Synthesize
Parametric Models & Constraints
Sketch & Desired Behavior

Fig. 21. Overview of the SKETCHIT system.

Fig. 21 illustrates SKETCHIT's implementation. Reverse engineering and generalization are implemented using generate and test. The qc-space generator produces candidate qc-space representations of the sketch and passes them to the simulator. Each candidate qc-space is a guess at what behavior each engagement pair should provide. The first candidate is an abstraction of the numerical c-space of the sketch; the rest are modifications of the first.

The simulator computes the overall behavior of each candidate, which the tester then compares to the desired behavior described in the state transition diagram. This process continues until the tester finds all qc-spaces that behave as desired.[9]

The synthesis process is implemented with two modules. The first selects a motion type for each component; the second selects a geometric implementation for each engagement

[9] Section 4 describes the range of candidates that the program considers.

pair from a library of interactions. Each library entry contains a pair of parameterized faces and constraints that ensure that the faces implement a specific kind of behavior. SKETCHIT assembles the parametric geometry and constraints from the library selections into a BEP-Model.

The bulk of SKETCHIT's effort is spent reverse engineering and generalizing the design. Because the program synthesizes new designs by using a library of interactions, the synthesis process is straightforward and computationally inexpensive.

4. The qc-space generator

The first step in reverse engineering and generalization is the generation of a qc-space from the sketch. The qc-space generator begins by computing the numerical c-space of the sketch, then abstracts each numerical cs-curve into a qcs-curve. The generator determines the qualitative slope of each qcs-curve by abstracting the slope of the corresponding numerical cs-curve: it computes the numerical slope of a straight line connecting the end points of the cs-curve and matches this to one of the eight curves in Fig. 19. The program gets the relative locations of the qcs-curves directly from the locations of the cs-curves in the numerical c-space diagram.

As with any abstraction process, moving from specific numerical curves to qualitative curves can introduce ambiguities. For example, in the candidate qc-space in Fig. 22 there is ambiguity in the relative location of landmark E (the abscissa value for the intersection between the push-pair curve and the pushrod-stop curve). This value is not ordered with respect to landmarks B and C (the abscissa values of the end points of the lever-stop and cam-follower curves in the hook-lever cs-plane). In qc-space E may be less than B, greater than C, or between B and C,[10] while in the numerical c-space E could be compared to B and C. When the generator encounters this kind of ambiguity, it enumerates all possible interpretations, passing each of them to the simulator.

Physically, landmark E denotes the configuration in which the lever is against the pushrod and the pushrod is against its stop; the ambiguity is whether in this particular configuration the lever is (a) to the left of the hook ($E < B$), (b) contacting the hook ($B < E < C$), or (c) to the right of the hook ($C < E$).

There are several reasons why SKETCHIT computes all possible relative locations of these landmarks, rather than taking the locations directly from the unambiguous numerical c-space. One reason is that it offers a means of generalizing the design: the original locations may be just one of the possible working designs; the program can find others by enumerating and testing all the possible relative locations. Said differently, in order to generate a wide range of design alternatives we want to generalize each cs-curve to the largest possible family of monotonic cs-curves. Ambiguity in the locations is the price we pay for generalizing single curves into families of curves.

A second, perhaps more interesting reason the program enumerates and tests all possible relative locations is because this enables it to compensate for flaws in the original sketch. These flaws arise from interactions that are individually correct, but whose global

[10] We do not consider the case where $E = B$ or $E = C$.

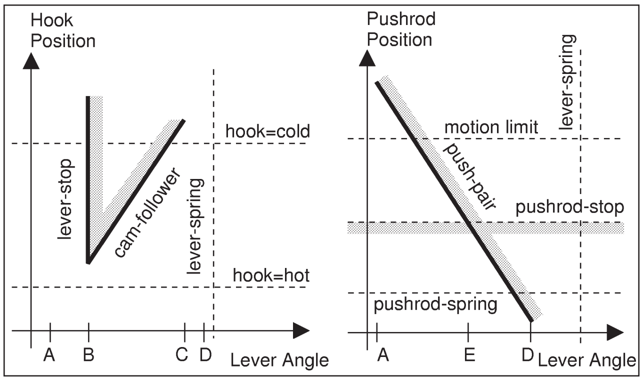


Fig. 22. A candidate qc-space for the circuit breaker. Only two of the three qcs-planes are shown. (The Pushrod-Hook qcs-plane provides no additional information.) The "hook = cold" and "hook = hot" curves are the neutral positions of the hook corresponding to its normal and over-heated temperatures. The "lever-spring" and "pushrod-spring" curves are the neutral positions of the springs attached to the lever and the pushrod. The "motion-limit" curve is the extent of the reset stroke. The remaining four curves are the interactions between the faces from Fig. 3. (Diagonal lines are drawn straight for convenience, but they can have any shape as long as they are monotonic.)

arrangement is incorrect. For example, in Fig. 3 the interaction between the lever and hook, the interaction between the pushrod and the lever, and the interaction between the pushrod and its stop may all be individually correct, but the pushrod-stop may be sketched too far to the left, so that the lever always remains to the left of the hook (i.e., the global arrangement of these three interactions prevents the lever from actually interacting with the hook). By enumerating possible locations for the intersection between the pushrod-stop and push-pair qcs-curves, SKETCHIT will correct this flaw.

As the generator abstracts the numerical c-space, it preserves four properties: the number of components (bodies, springs, actuators, and engagement pairs); the qualitative slopes of the cs-curves; the relative locations of the cs-curves; and the intersections between the cs-curves, because we take these properties as the intended meaning of the sketch. Consequently, as the program generates alternative designs, it holds these properties constant.[11] The set of candidate qc-spaces the generator produces for a sketch consists of all possible interpretations of the ambiguities inherent in the abstraction. The simulator and tester identify which of these interpretations produce the desired behavior.

4.1. Preserving intersections

4.1.1. Intersections of finite qcs-curves

If two finite cs-curves intersect in c-space, we want to ensure that the corresponding qcs-curves intersect in qc-space. If one of the cs-curves is horizontal and the other vertical, the qcs-curves will naturally intersect. However, if one or both of the cs-curves is diagonal, explicit constraints are necessary to ensure intersection.

[11] We could have used another definition by choosing to preserve more or fewer properties from the numerical c-space. By preserving more properties we would generate a narrower range of design alternatives, and vice versa.

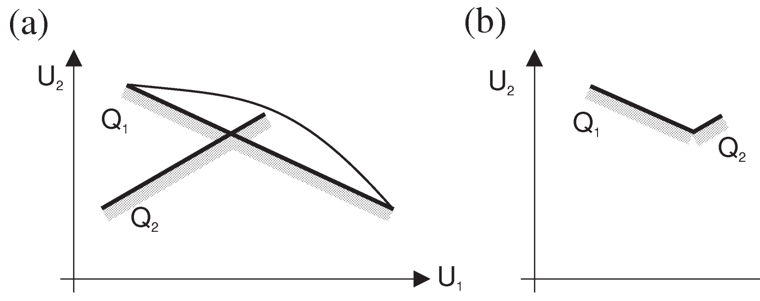


Fig. 23. (a) Depending on their particular shapes, the qcs-curve may or may not intersect. (b) When the useless ends of Q_1 and Q_2 are removed, the curves intersect at a common end point.

Consider the intersection of the two diagonal qcs-curves Q_1 and Q_2 shown in Fig. 23(a). Q_1 and Q_2 can have any shape as long as they are monotonic. Depending on the actual shape, the curves may or may not intersect. For example, if Q_1 curves up (thin line) and Q_2 is straight, the curves will miss each other.

We use a trimming operation to ensure that finite qcs-curves intersect. When two such curves intersect, one end of each curve will be useless because it lies in the blocked space of the other curve (i.e., during simulation the trajectory in qc-space cannot reach that part of the curve). In the example in Fig. 23(a), the portions of the curves below the intersection point are the useless portions. If we discard them there will be no change in the behavior of the device because the device configuration can never reach these portions. However because the two curves now have a common end point, they are guaranteed to intersect (Fig. 23(b)).[12]

4.1.2. Ensuring that finite qcs-curves do not intersect

If two finite cs-curves do not intersect in c-space, we want to place constraints on the corresponding qcs-curves to ensure that they do not intersect in qc-space. Explicit constraints are necessary only when the bounding boxes of the curves overlap, because otherwise there is no possibility of intersection. Diagonal curves are the only kinds of curves that can have overlapping bounding boxes and still not intersect. Hence, our problem reduces to the single case of diagonal curves with overlapping bounding boxes.

We use the example in Fig. 24 to show how SKETCHIT constructs non-intersection constraints. End point (A, B) of curve Q_1 lies inside the bounding box of the other curve, Q_2. SKETCHIT must ensure that this point lies above and to the left of Q_2. To facilitate this, SKETCHIT labels two special points, (A, A′) and (B′, B), on Q_2. These points are established by passing a vertical line and a horizontal line through point (A, B) and intersecting them with Q_2. Using these points, the non-intersection constraints are expressed as: $\mathrm{A}' < \mathrm{B}$ and $\mathrm{B}' > \mathrm{A}$. These constraints are then added to the BEP-Model.

[12] In the examples we tried, none of the curves required trimming, and hence we did not implement a subroutine to trim qcs-curves. However, this is a straightforward extension of the program.

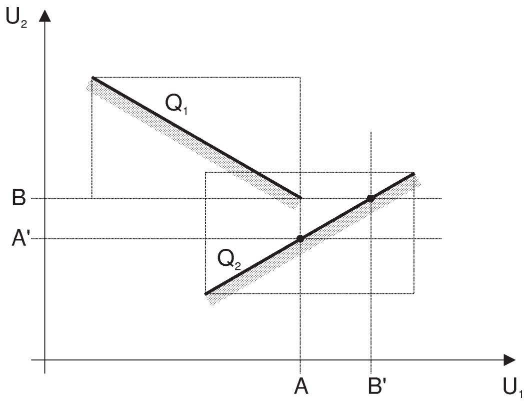


Fig. 24. Two non-intersecting, finite qcs-curves with intersecting bounding boxes.

4.1.3. Intersections with infinite boundaries

If a pair of infinite boundaries intersect in c-space, they will naturally intersect in qc-space. Hence SKETCHIT need not perform any special operations to preserve the intersections during abstraction.

4.2. Enumerating possible locations of intersection points

Even though SKETCHIT preserves intersection points, the locations of intersections may still be ambiguous in qc-space. For example, the push-pair curve and pushrod-stop curve in Fig. 22 are guaranteed to intersect (because the latter curve is an infinite boundary), but as we described above, the location is ambiguous.

The efficient way to handle this kind of ambiguity is to wait and see if the simulator needs the precise location of the intersection. It will need this if the trajectory through qc-space enters the bounding box of the diagonal curve. When and if this happens the simulator can enumerate all of the possible locations of the intersection point and branch to consider each possibility. If the simulator never requires information about the location (i.e., the trajectory never enters the bounding box), the precise location has no influence on the overall behavior of the device.

For example, if the trajectory reaches the bounding box of the push-pair qcs-curve in Fig. 22, the simulator must enumerate the possible locations of the intersection between this qcs-curve and the infinite boundary for the pushrod-stop. As described above, there are three possible locations ($A < E < B$, $B < E < C$, and $C < E < D$) and the simulator would have to branch to consider all three. In effect, the simulator would be specializing the current candidate qc-space into three new candidates.

If the trajectory never reaches the bounding box, the program need not branch because the set of new, specialized candidates will all produce the same behavior. Hence, in this case, by leaving the location ambiguous, the program would be using a single simulation to compute the behavior of a set on unambiguous candidate qc-spaces (i.e., the set of new, specialized qc-spaces).

4.3. Implementation notes

The previous sections described efficient methods for preserving intersections and enumerating possible locations of intersection points. The current version of the program actually uses simpler techniques, which although inefficient, are much easier to implement.

The program extends horizontal and vertical lines through the end points of each qcs-curve (just as we did in Fig. 24) and intersects these with each of the diagonal qcs-curves. The program creates a new landmark for each of the intersection points. Similarly, the program creates a new landmark for each intersection between an infinite boundary and a diagonal qcs-curve.

The coordinates of an intersection point are defined by one old landmark (i.e., a landmark that denotes the end point of a finite qcs-curve or the axis crossing of an infinite boundary) and one new landmark. Thus, the intersection points for any particular qcs-curve are partially ordered: those points whose x-coordinate is an old landmark form one ordered set, those whose y-coordinate is an old landmark form another. The program enumerates all possible relative locations of the intersections by simply enumerating all possible ways of interleaving the elements of these two sets (while, of course, maintaining the two partial orderings).

After enumerating all possible locations of the intersection points, the program filters out those choices that do not satisfy the non-intersection constraints described above. The program then simulates and tests all of the remaining choices. Each of these choices is completely unambiguous.

4.3.1. Periodic boundary conditions

As Section 2.1 described, a pairwise qc-space will wrap around into a cylinder if one of the bodies rotates more than a full revolution. During abstraction, the generator must preserve the periodicity of the cylinder. To do this, it first constructs boundaries in the numerical c-space at 0 and 2π and computes the intersections between them and the cs-curves. The program then cuts the cylinder along the boundaries (which coincide on the cylinder) and unrolls it into a plane. Finally, the program abstracts this plane, including the boundaries and their intersections, into a qcs-plane.

During simulation, when the trajectory through the plane reaches one of the boundaries, the simulator simply moves the trajectory to the other boundary (i.e., when the angle reaches 2π the simulator changes the angle to 0, or vice versa).

The yoke and rotor device which we will discuss in Section 9 provides an example of a cylindrical c-space. Fig. 36 shows the flattened out version of the qc-space SKETCHIT obtains by adding boundaries and unrolling the cylinder. Curves A1 and B2 intersect the boundaries, hence one piece of each of these curves is just above the 0 boundary and another piece is just below the 2π boundary in the flattened out version.

4.3.2. Repairing flaws in the sketch

By enumerating the possible locations of intersection points, the current SKETCHIT system can repair a limited range of flaws in the original sketch (i.e., incorrect global arrangement of interactions that are all individually correct). We are continuing to work on techniques for repairing more serious kinds of flaws such as individual interactions that are locally (as well as globally) defective.

Because there are only two properties in qc-space that matter—the relative locations and the qualitative slopes of the qcs-curves, to repair a sketch, even one with serious flaws, the task is to find the correct relative locations and qualitative slopes for the qcs-curves.

We can do this using the same generate and test paradigm described earlier. In this case, our search space is all possible relative locations (i.e., all possible orderings of the curve end points) and all possible choices of qualitative slopes. Because there are often many changes to the geometry that map to a single change in qc-space, our search space is much smaller than the space of possible modifications to the geometry. Repairing a design by repairing its qc-space is thus more efficient than directly repairing the geometry.

However, for realistic designs, even this search space is far too large for exhaustive search. Thus, we are exploring several ways to minimize search. We are, for example, exploring the use of debugging rules that examine *why* a particular qc-space fails to produce the correct behavior, based on its topology. The desired behavior of a mechanical device can be described as a desired trajectory through its qc-space. The topology of the qc-space can have a strong influence on whether the desired trajectory (and the desired behavior) is easy, or even possible. For example, the qc-space may contain a funnel-like topology that "traps" the device, preventing it from traversing the desired trajectory. If we can diagnose these kinds of failures, we may be able to generate a new qc-space by judicious repair of the current one.

Another possible way to reduce search is by identifying those parts of the qc-space that are likely to be correct, so that the repair effort can be focused on the other parts. For example, we could simulate each arc of the state transition diagram individually to see which arcs produce the desired state transitions. If a particular arc does produce the desired transition, the parts of the qc-space used in that transition are likely to be correct. Conversely, if a particular arc does not produce the desired state transition, the parts of the qc-space used in that transition are likely to need repair.

5. Simulator

Qc-space represents the device's kinematics, that is, it describes all possible positions the device's parts can occupy. We use a qualitative simulator to determine which sequences of positions (motions) the parts will actually exhibit in response to the applied inputs specified with the state transition diagram. This section provides a brief, high-level overview of our simulator.

The simulator operates using the particle metaphor described in Section 2.1.2. It begins by computing the net force on each body. In an inertia-free world, the velocity is in the direction of the net force and continues until some event (e.g., a collision) changes the nature of the forces.[13] When this happens the simulator stops, recomputes the forces, then continues simulating.

[13] With inertia, the acceleration is in the direction of the net force, but the velocity need not be.

Each possible kind of event corresponds to the trajectory in qc-space reaching or leaving a boundary—either the boundary between free and blocked space (i.e., a qcs-curve) or a boundary representing a spring neutral position or a motion limit of an actuator. Hence, to determine what events happen next, the simulator must examine the trajectory through qc-space.

We make a number of assumptions about devices that greatly simplify the task of reasoning about trajectories. We assume that qcs-curves are monotonic, motion is inertia-free, contacts are frictionless, and collisions are inelastic. As a result, until an event occurs, the motion of a body remains either strictly positive, strictly negative, or zero. Consequently, all trajectories in qc-space are monotonic over any given time step.

We assume also that devices are fixed-axis and as a result we can examine the trajectory through a multi-dimensional qc-space by examining the trajectories through 2D projections of the space, i.e., the qcs-planes. We compute the next event by first determining which events would happen if the trajectory in each qcs-plane continued until an event occurs in that plane, and we then use constraint propagation to determine which of the events predicted by the individual planes can happen first.

Because the simulation is qualitative, ambiguities can arise concerning what will happen next; our simulator produces an envisionment by determining all possible sequences of events.

As mentioned above, the simulator begins each step of simulation by computing the resultant of all the forces applied to a body in order to determine its motion until the next event. Because qc-space is qualitative, the program must use a qualitative representation for forces. The obvious approach, representing a force as a qualitative vector, results in a significant amount of ambiguity in the force sums: each component of a qualitative vector is a qualitative scalar and a sum of qualitative scalars is subject to ambiguity. Because a qualitative vector has many scalar components, there are many ways for a sum of qualitative vectors to be ambiguous. Hence, we developed special techniques for representing and reasoning about forces.

First, we represent a force by its projection on the degree of freedom of the body to which it is applied. Because this projection is the only component of the force that has any effect on the motion of the body, this simplification introduces no inaccuracies. The advantage of this simplification is that it greatly reduces ambiguity in force sums.

Second, we represent engagement forces (the forces that engagement faces apply to each other) by the type of constraint they impose. Consider, for example, the three blocks in Fig. 25. The spring pushes block A to the right, the actuator pushes block C to the left. In SKETCHIT's world, all actuators are assumed to be motion sources, that is, they assign position as a function of time. Block B, the block in the middle, experiences two engagement forces, one from A and one from C. Because the forces are in opposite directions, the qualitative sum of these forces is ambiguous.

However, we know that B will move to the left. Why is this? We know that the force C applies to B is whatever force is necessary for C to achieve its assigned motion. We call this kind of engagement a "motion constrained engagement" because it constrains the motion of the body to which it is applied.

According to Hooke's law, the spring's deflection determines the magnitude of the spring's force on A. In an inertia-free world A will transmit the spring force to B. Thus,

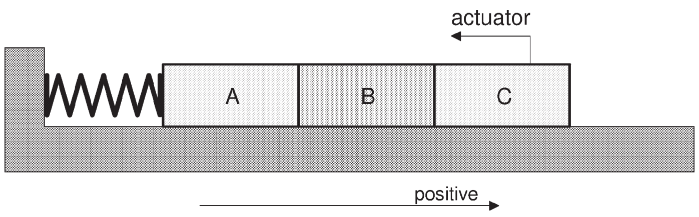


Fig. 25. Three blocks sliding on a frictionless, horizontal surface.

A applies a force of known magnitude to B. In contrast to a motion constrained engagement force which assigns a known motion, this kind of force assigns a known magnitude. We call this type of force a "compliant engagement force" because it has a known magnitude in the same way that a compliant member (e.g., a spring) produces a force of known magnitude.

One of our basic principles is that a motion constrained engagement overpowers a compliant engagement. Hence, B moves to the left. In our experience, this principle has proven very useful in resolving ambiguities in force sums.

A complete description of our simulator, including our other techniques for reducing ambiguities, can be found in [43] and [46].

6. The tester

The tester compares the simulated behavior of the candidate qc-spaces to the desired behavior specified with the state transition diagram, accepting those candidates that provide the desired behavior.

The simulator produces an envisionment by computing all possible motions the device can exhibit in response to the applied inputs. If all of the branches pass through the desired sequence of states specified by the state transition diagram, the candidate will provide only the desired behavior. On the other hand, if none of the branches pass through the desired sequence of states, the candidate is incapable of providing the desired behavior. If some branches do and others do not, the candidate will produce other behaviors in addition to the desired behavior.[14] The tester accepts those candidates that produce only the desired behavior.

7. The motion type selector

The motion type selector's task is to select a motion type of either rotation or translation for each part in the device. If the qc-space coordinate of a body is non-periodic, the body's

[14] These device will provide the desired behavior only for specific choices of masses, springs, and actuators. We could accept these designs and then use numerical simulation to verify that they produce the desired behavior once these choices have been made. Hence, we would use qualitative simulation for conceptual design and progress to more precise numerical techniques (which are computationally more expensive) as more of the design details are selected. An alternative approach would be to construct additional constraints on the masses, springs, and actuators to ensure that only the desirable branches of the simulation are possible. These constraints would be added to the BEP-Models computed from the qc-space.

motion type can be either translation or rotation, but if rotation is selected the body will rotate less than a full revolution. If the body's qc-space coordinate is periodic (i.e., wraps around to form a cylinder) the only possible motion type is rotation. In this case the body will rotate through full revolutions.

This distinction can be cast in more physical terms. If a part that rotates through full revolutions turns far enough in a single direction, it will return to the position from which it started, without ever having to reverse its direction of motion. The only way that a translating part can return to the place from which it started is if the part reverses its direction of motion. Hence, a translating part cannot replace a rotating part that turns through full revolutions.

By selecting new motion types for each body, SKETCHIT can generate a rich assortment of new designs, as illustrated in Fig. 8. The new designs are completely consistent with the desired behavior because they produce the sequence of engagements given in the state transition diagram.

8. Interaction library

The general task of translating from c-space to geometry is intractable [2]. However, translating from qc-space to geometry is tractable because qc-space is carefully designed to be constructed from a small number of basic building blocks (qcs-curves). There are 40 such building blocks. [15]

Because there is only a small number of basic building blocks, we were able to construct a library of implementations for each of them. To translate a qc-space to geometry, the program simply selects from the library an implementation for each qcs-curve.

Each library entry contains a pair of parameterized faces and a set of constraints that ensure the faces implement a monotonic cs-curve with the desired qualitative slope and the desired choice of blocked space. Each library entry also contains algebraic expressions for the coordinates of the cs-curve end points.

For example, Fig. 26 shows a library entry for qcs-curve F in Fig. 19, for the case in which q_1 is rotation and q_2 is translation (i.e., rotation in the negative direction causes translation in the negative direction). For the corresponding qcs-curve to be monotonic, have the correct slope, and have blocked space on the correct side, the following ten constraints must be satisfied (see Appendix A for the derivation):

$$w > 0 \tag{1}$$

$$L > 0 \tag{2}$$

$$h > 0 \tag{3}$$

[15] The origin of 32 of these can be seen by examining Fig. 19: there are four choices of qualitative slope; for each qualitative slope, there are two choices for blocked space; and the qc-space axes q_1 and q_2 can represent either rotation or translation. The remaining 8 building blocks represent interactions of rotating or translating bodies with stationary bodies. These interactions produce horizontal and vertical qcs-curves that are infinite versions of curves A, C, E, and G in Fig. 19. The number 8 comes from the fact that for each of these four types of infinite qcs-curves, one of the interacting bodies can either rotate or translate while the other body is fixed.

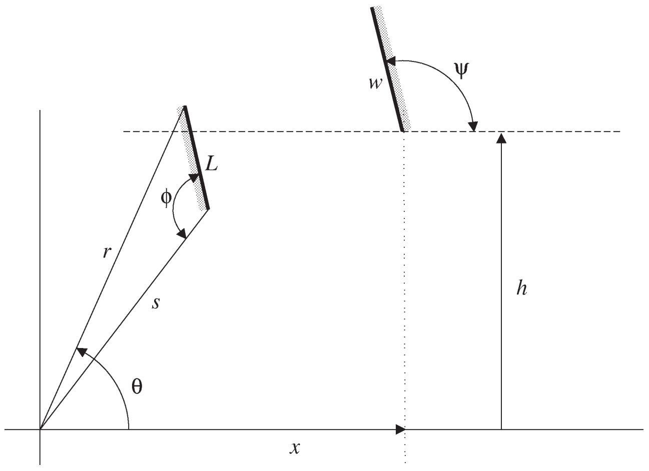


Fig. 26. Library entry F-1: an implementation for curve F (“cam and offset follower”). The two faces are shown as thick lines. The rotating face rotates about the origin; the translating face translates horizontally. θ, measured positive counterclockwise, is the angle of the rotor; x, measured positive to the *left*, is the position of the slider.

$$r = \left(s^2 + L^2 - 2sL\cos(\phi)\right)^{1/2} \tag{4}$$

$$r > h \tag{5}$$

$$s < h \tag{6}$$

$$\phi > \pi/2 \tag{7}$$

$$\arccos\left(\left(L^2 + r^2 - s^2\right)/(2Lr)\right) + \arccos(h/r) < \pi/2 \tag{8}$$

$$\phi \leqslant \pi \tag{9}$$

$$\psi < \arcsin(h/r) + \pi/2 \tag{10}$$

$$\psi > 0 \tag{11}$$

The coordinates of the cs-curve end points are: [16]

$$\theta_1 = \arcsin(h/r) \tag{12}$$

$$x_1 = -r\cos(\theta_1) \tag{13}$$

$$\theta_2 = \pi - \arcsin(h/r) \tag{14}$$

$$x_2 = r\cos(\theta_1) \tag{15}$$

Fig. 27 shows a second way to generate qcs-curve F, using the constraints:

$$L > 0 \tag{16}$$

$$h_1 > 0 \tag{17}$$

[16] (θ_1, x_1) is the lower left end of the curve. x is measured positive to the *left* in Fig. 26.

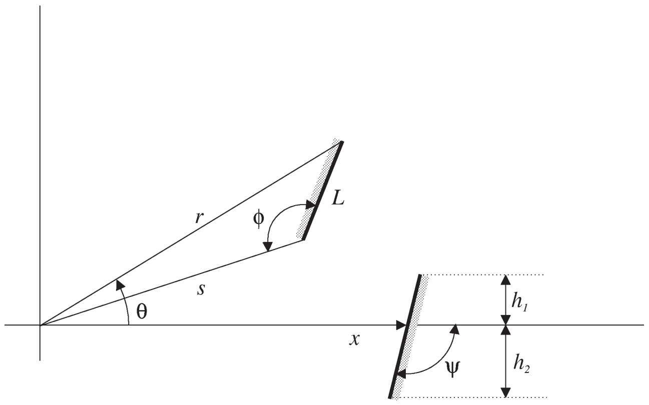


Fig. 27. Library entry F-2: an implementation for curve F (“cam and centered follower”). The two faces are shown as thick lines. The rotating face rotates about the origin; the translating face translates horizontally. θ, measured positive counterclockwise, is the angle of the rotor; x, measured positive to the *left*, is the position of the slider.

$$h_2 > 0 \tag{18}$$

$$\psi > \pi/2 \tag{19}$$

$$\psi < \pi \tag{20}$$

$$\phi > \pi/2 \tag{21}$$

$$\phi < \pi \tag{22}$$

$$r = \left(s^2 + L^2 - 2sL\cos(\phi)\right)^{1/2} \tag{23}$$

$$s > h_1 \tag{24}$$

$$0 > r/\tan(\psi) + h_2/\sin(\psi) \tag{25}$$

The coordinates of the cs-curve end points are: [17]

$$\theta_1 = -\arcsin(h_2/r) \tag{26}$$

$$x_1 = -r\cos(\theta_1) + h_2/\tan(\psi) \tag{27}$$

$$\theta_2 = \arcsin(h_1/s) + \arccos\left(\left(s^2 + r^2 - L^2\right)/(2sr)\right) \tag{28}$$

$$x_2 = -s\cos(\arcsin(h_1/s)) - h_1/\tan(\psi) \tag{29}$$

The slider in Fig. 26 does not pass through the pivot, the one in Fig. 27 does. In the first design the motion of the slider is approximately parallel to the motion of the rotor, while in the second the motion of the slider is approximately perpendicular to the motion of the rotor. The first of these is a cam with offset follower, the second is a cam with centered follower. The two designs thus represent qualitatively different implementations for the same qcs-curve.

[17] (θ_1, x_1) is the lower left end of the curve. x is measured positive to the *left* in Fig. 27.

To generate a BEP-Model for the sketch, the program first selects from the library an implementation for each qcs-curve.[18] For each selection it creates new instances of the parameters and transforms the coordinate systems to match those used by the actual components. The relative locations of the qcs-curves in qc-space are turned into constraints on the end points of the cs-curves. SKETCHIT assembles the parametric geometry fragments and constraints of the library selections to produce the parametric geometry and constraints of the BEP-Model.

Our library contains geometries that use flat faces,[19] although we have begun work on using circular faces. We have at least one library entry for each of the 40 kinds of interactions and are continuing to generate new entries.

By selecting different library entries for a given qcs-curve, SKETCHIT is able to produce different BEP-Models (i.e., different families of designs). For example, Fig. 5 shows a BEP-Model SKETCHIT generates by selecting library entry F-2 for the cam-follower qcs-curve. Fig. 9 shows a solution to a different BEP-Model SKETCHIT generates by instead using F-1. As these examples illustrate, the designs generated by selecting different library entries can encompass a wide variety of design solution.

8.1. *Constructing a BEP-Model*

This section describes in more detail how SKETCHIT uses the library to construct a BEP-Model. We illustrate this process by an example, showing how SKETCHIT constructs a BEP-Model for the circuit breaker qc-space in Fig. 28.

SKETCHIT can construct many different BEP-Models for this particular qc-space. For example, if the motion type selector chooses translation for the motion types of the hook and pushrod and rotation for the lever, the program will derive four different BEP-Models. Here we show how it constructs just one of the four, the one corresponding to the geometry in Fig. 9.

To construct this particular BEP-Model, SKETCHIT uses library entry F-1 (Fig. 26) to implement the cam-follower qcs-curve, D-2 (similar to Fig. 27 but using the coordinate transformations in Appendix 12) for the push-pair, TRS-1 (Fig. 29) for the lever-stop, and FSS-1 (Fig. 30) for the pushrod-stop.

To use library entry F-1 for the cam-follower, SKETCHIT creates new instances of the parameters and instantiates the constraints as shown in Fig. 31. SKETCHIT creates new parameter instances by appending a unique integer to the end of each parameter name. For example, `L_7` is a new instance of the parameter L.

The first four constraints in Fig. 31 assign values to the end point landmarks of the cam-follower qcs-curve using Eqs. (12)–(15).[20] These constraints include a pair of parameters,

[18] The neutral positions of springs and the motion limits of actuators produce qcs-curves, but do not describe interacting faces, and hence require no geometric implementation.

[19] Circular faces are used when rotors act as stops.

[20] To be rigorous we must constrain the landmarks of rotating bodies to be between 0 and 2π. This is true for bodies that are intended to rotate less than a full revolution as well as for bodies that are intended to rotate more than a full revolution. In the former case, constraining the landmarks to be between 0 and 2π ensures that the implementation will actually rotate less than a full revolution. In the latter case, the device's qc-space is actually a flattened out representation of a cylinder or torus (see Section 4.3.1). In this flattened out representation all

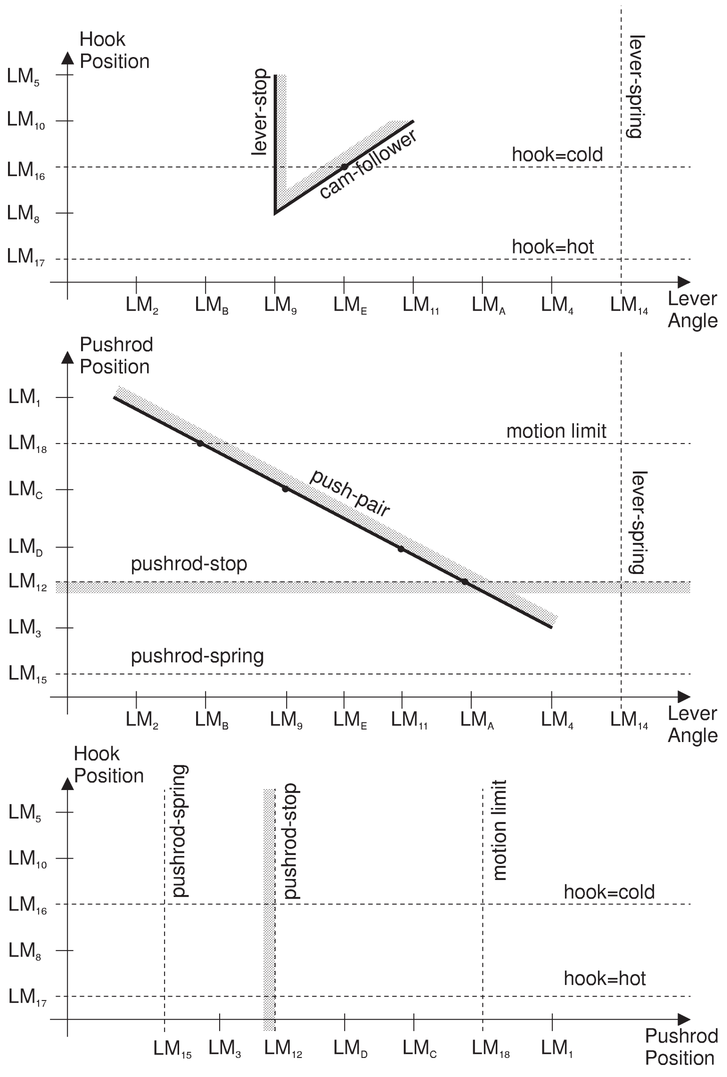
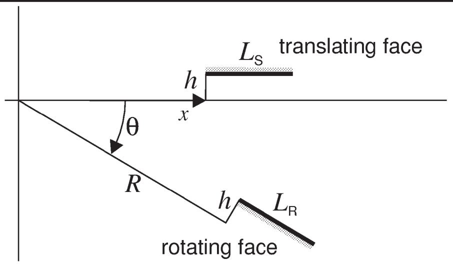


Fig. 28. A circuit breaker qc-space that provides the desired behavior.

`off_1` and `off_2`, that allow the faces to be attached to the hook and lever at arbitrary locations (subject, of course, to the other constraints).

SKETCHIT repeats this process for the push-pair, the lever-stop, and the pushrod-stop producing the constraints in Fig. 32.

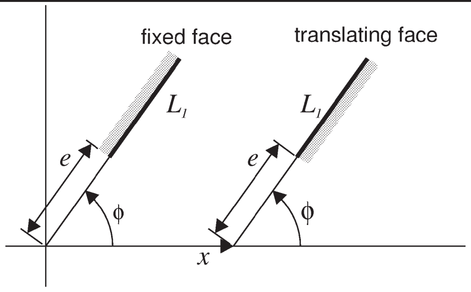
landmarks are explicitly constrained to be between the boundaries at 0 and 2π. For the circuit breaker example, SKETCHIT did not explicitly constrain the landmarks of the lever to be between 0 and 2π, but it did do this in the yoke and rotor example.



Constraints:
$L_R, L_S, R > 0$

End Points:
$\theta_1 = 0$
$x_1 = R - L_S$
$\theta_2 = 0$
$x_2 = R + L_R$

Fig. 29. **Left:** library entry TRS-1: geometry used to implement qcs-curve C in Fig. 19, for the case in which q_1 is rotation and q_2 is translation (i.e., a translating rotor-stop). The position of the rotor is measured positive *counterclockwise* with angle θ. The translating face translates horizontally with position x. **Right:** constraints on the geometry to ensure a monotonic cs-curve with correct qualitative slope and with blocked space on the correct side; the end points of the cs-curve.



Constraints:
$L_l, e > 0$
$0 < \phi < \pi$

Axis Crossing:
$x = 0$

Fig. 30. **Left:** library entry FSS-1: geometry used to implement an infinite version of qcs-curve A in Fig. 19, for the case in which q_2 is translation and q_1 is either rotation or translation (i.e., a stationary slider-stop). The translating face translates horizontally with position x. **Right:** constraints on the geometry to ensure a monotonic cs-curve with correct qualitative slope and with blocked space on the correct side; the position at which the infinite boundary crosses the coordinate axis.

Next, SKETCHIT develops expressions for the landmarks for the points of intersection between qcs-curves. Consider the point defined by the intersection of the horizontal “hook = cold” line and the cam-follower curve in the top of Fig. 28. The location of this intersection determines the value of landmark $\mathtt{LM}_E$. SKETCHIT instantiates this constraint with the statement “`(LM_E = (INTERSECT (HOOK = LM_16) CAM-FOLLOWER))`”, indicating that the value of landmark $\mathtt{LM}_E$ is defined by the intersection between the horizontal line located at landmark $\mathtt{LM}_{16}$ and the cam-follower cs-curve. Hence, to compute the value of $\mathtt{LM}_E$ the constraint solver (the program that evaluates the BEP-Model) must compute the intersection between the horizontal line and the cs-curve.

There are 5 intersection points in Fig. 28 (shown as dots). Each one produces a constraint similar to that defining $\mathtt{LM}_E$. Fig. 33 shows these constraints.

```
LM_9 = ASIN (H_4 / R_6) + off_1
LM_8 = - (R_6 * COS (ASIN (H_4 / R_6))) + off_2
LM_11 = 180 - ASIN (H_4 / R_6) + off_1
LM_10 = - (- R_6 * COS (ASIN (H_4 / R_6))) + off_2
W_8 > 0
L_7 > 0
H_4 > 0
R_6 = SQRT (S_5 ^2 + L_7 ^2 - 2* S_5 * L_7 * COS (PHI_9))
R_6 > H_4
S_5 < H_4
PHI_9 > 90
ACOS (H_4 / R_6)
 + ACOS((L_7 ^2 + R_6 ^2 - S_5 ^2) / (2 * L_7 * R_6)) < 90
PHI_9 <= 180
PSI_10 < ASIN (H_4 / R_6) + 90
PSI_10 > 0
```

Fig. 31. The constraints for the portion of the BEP-Model describing the cam-follower interaction.

To complete the constraints of the BEP-Model, SKETCHIT instantiates the constraints on the landmark orderings as shown in Fig. 34.[21]

Figs. 31–34 constitute the complete set of constraints for this BEP-Model. The model also contains parametric geometry. SKETCHIT produces this by assembling the parametric geometry from each of the library selections (i.e., the geometry in Figs. 26, 27, 29, and 30).

Before assembling the geometry, SKETCHIT must first establish coordinate systems for each of the components as shown in Fig. 35. There are two coordinate frames associates with each component: one is fixed to the body, the other is fixed to ground. The motion type of a body determines the relative motion between its two frames. For example, the X_L – Y_L frame, which is attached to the lever, rotates relative to the stationary $\mathsf{X}_{\mathsf{L-G}}$ – $\mathsf{Y}_{\mathsf{L-G}}$ frame. The former frame represents the lever, the latter represents the lever's pivot (the origins of the two frames remain coincident).[22] Dimension x_L measures the lever's angle. The coordinate systems of the pushrod and hook are defined similarly, except that the body fixed frames translate rather than rotate.

These pairs of coordinate frames, which represent the locations of the components, can initially be placed arbitrarily in the plane. The constraints of the BEP-Model will ensure that they assume positions that allow the device to provide the correct behavior. For sake

[21] The generator assigns a total ordering to the landmarks of the end points and axis crossings (these landmarks have a numeric subscript in Fig. 28). It also assigns a total ordering to the landmarks of the intersection points of each individual diagonal qcs-curve (these landmarks have an alphabetic subscript in Fig. 28). When SKETCHIT instantiates the constraints imposed by the various totally ordered sets, it may instantiate redundant constraints. For example, the constraints `(LM_E < LM_11)` and `(LM_9 < LM_E)` come from the intersection points of cam-follower curve while the constraint `(LM_9 < LM_11)` comes form the ordering of the end point landmarks. However, this later constraint is subsumed by the other two. Fortunately, this kind of redundancy causes no difficulties.

[22] The "L" in the subscript stands for lever, the "G" stands for "guide". (We use the term guide to refer to both the pivot of a rotating body and the axis of translation of a translating body.)

```
;; push-pair
LM_2 = - (ASIN (H1_30 / S_32)
          + ACOS ((S_32 ^2 + R_33 ^2 - L_34 ^2) / (2* S_32 * R_33)))
       + off_3
LM_1 = - S_32 *COS (ASIN (H1_30 / S_32)) - H1_30 / TAN (PSI_36) + off_4
LM_4 = ASIN (H2_31 / R_33) + off_3
LM_3 = - R_33 * COS (ASIN (H2_31 / R_33)) + H2_31 / TAN (PSI_36) + off_4
L_34 > 0
H1_30 > 0
H2_31 > 0
PSI_36 > 90
PSI_36 < 180
PHI_35 < 180
PHI_35 > 90
R_33 = SQRT (S_32 ^2 + L_34 ^2 - 2* S_32 * L_34 * COS (PHI_35))
S_32 > H1_30
0 > R_33 / TAN (PSI_36) + H2_31 / SIN (PSI_36)

;; lever-stop
LM_9 = off_5
LM_8 = R_21 - LS_19 + off_6
LM_5 = R_21 + LR_18 + off_6
LS_19 > 0
LR_18 > 0
R_21 > 0

;; pushrod-stop
LM_12 = off_7
E_3 > 0
PHI_2 > 0
PHI_2 < 180
L1_1 > 0
```

Fig. 32. The constraints for the portions of the BEP-Model describing the push-pair, lever-stop, and pushrod-stop interactions.

```
LM_E = (INTERSECT (HOOK = LM_16) CAM-FOLLOWER)
LM_B = (INTERSECT (PUSHROD = LM_18) PUSH-PAIR)
LM_C = (INTERSECT (LEVER = LM_9) PUSH-PAIR)
LM_D = (INTERSECT (LEVER = LM_11) PUSH-PAIR)
LM_A = (INTERSECT (PUSHROD = LM_12) PUSH-PAIR)
```

Fig. 33. The intersection points for the circuit breaker qc-space. `LM_C` and `LM_D` are artifacts of the way SKETCHIT computes intersections between diagonal qcs-curves and infinite boundaries (see Section 4.3).

```
LM_18 < LM_1        LM_C < LM_18        LM_D < LM_C
LM_12 < LM_D        LM_3 < LM_12        LM_15 < LM_3
LM_12 < LM_18       LM_4 < LM_14        LM_A < LM_4
LM_11 < LM_A        LM_E < LM_11        LM_9 < LM_E
LM_B < LM_9         LM_2 < LM_B         LM_9 < LM_11
LM_2 < LM_9         LM_11 < LM_4        LM_10 < LM_5
LM_16 < LM_10       LM_8 < LM_16        LM_17 < LM_8
```

Fig. 34. The landmark orderings for the circuit breaker qc-space.

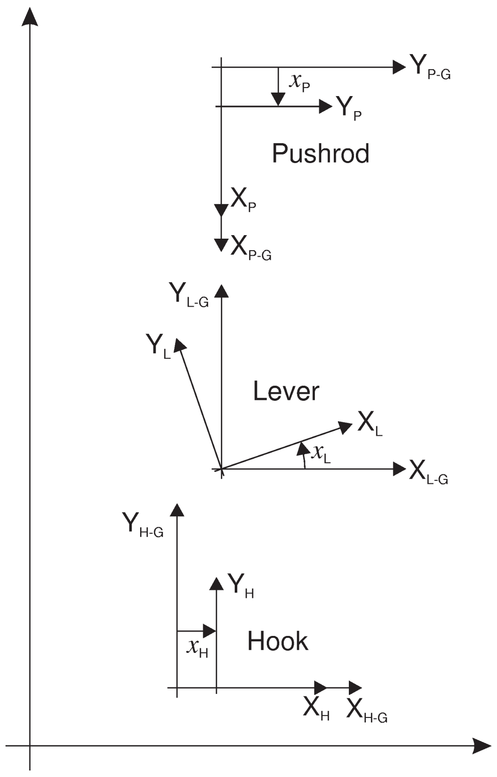


Fig. 35. Pairs of coordinate frames defining the locations of the components in the circuit breaker.

of presentation, Fig. 35 shows the coordinate frames in positions that are likely to provide the correct behavior (i.e., satisfy the constraints of the BEP-Model).

After establishing the coordinate systems of the components, SKETCHIT is ready to assemble the fragments of parametric geometry from the library selections. This geometry is defined with respect to local coordinate systems contained in each library entry. Hence, to assemble the geometry, SKETCHIT must transform from the local, library coordinate systems to the coordinate systems of the components.

8.2. Periodic boundary conditions

To facilitate simulation, SKETCHIT flattens out cylindrical qc-spaces into planes with boundaries at 0 and 2π. During synthesis SKETCHIT must in effect reconstruct the cylindrical qc-space from the flattened representation.

The reconstruction process focuses on the qcs-curves that cross the boundaries. In the flattened out representation, the boundary-crossing qcs-curves are split into two pieces: one piece just above the 0 boundary and another just below the 2π boundary. To facilitate synthesis, SKETCHIT must join these pieces back together.

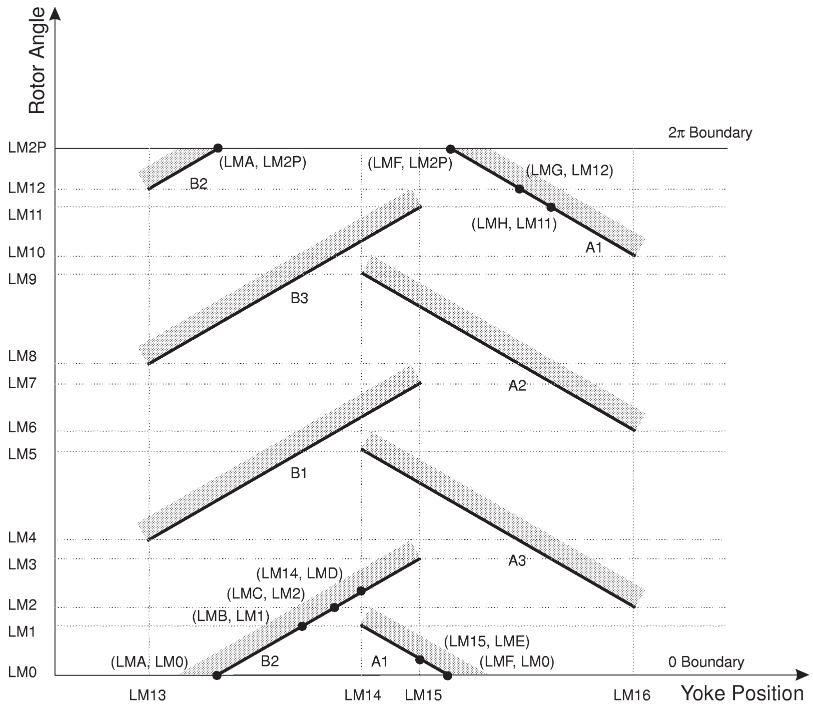


Fig. 36. The qc-space for the yoke and rotor flattened to produce a plane. The labels on the qcs-curves indicate the names of the interacting faces. For example, curve A1 is the interaction between face A on the yoke and face 1 on the rotor (see Fig. 39(a)). Because curves A1 and B2 cross the boundaries, one piece of each of these curves is just above the 0 boundary and another piece is just below the 2π boundary. The small black circles with accompanying coordinate tuples are intersection points (not all intersection points are labeled). Although the qcs-curves are drawn as straight lines, they can have any shape as long as they are monotonic.

SKETCHIT joins the two pieces of a split curve by subtracting 2π from the ordinates of the end points and intersection points of the piece of the curve near the 2π boundary (here we assume that the ordinate is the dimension that wraps around to form a cylinder). For example, Fig. 36 shows the flattened qc-space for the yoke and rotor from the next section, and Fig. 37 shows the result of joining the boundary-crossing curves back together.

To construct a BEP-Model, SKETCHIT selects appropriate library entries for the re-joined boundary-crossing qcs-curves. Because the remaining qcs-curves lie entirely between the boundaries of the flattened out representation, SKETCHIT can directly select library entries for them without any additional effort.

9. Using the BEP-model to refine a concept

As we have noted, the constraints in each BEP-Model represent the range of values that the geometric parameters can take on and still provide the behavior originally specified.

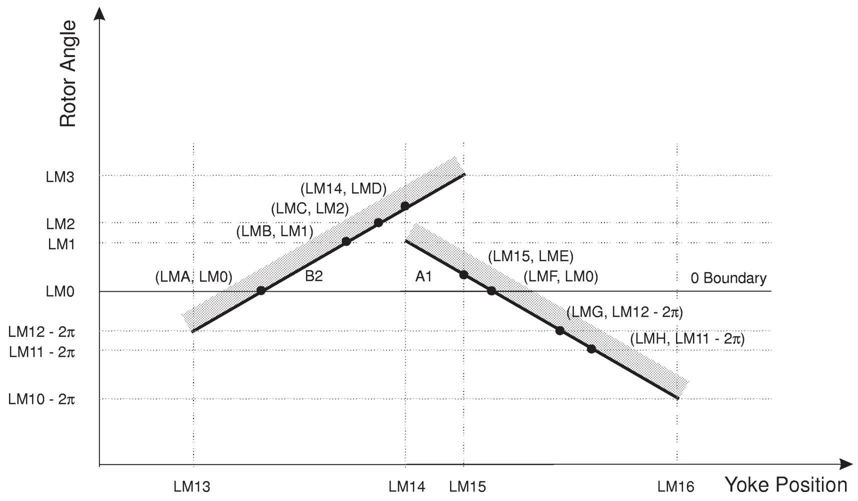


Fig. 37. A reconstruction of the boundary-crossing qcs-curves from the yoke and rotor qc-space. The pieces of the boundary-crossing curves near the 2π boundary in Fig. 36 have been moved below the 0 boundary by subtracting 2π from the ordinates of their end points and intersection points. The small black circles with accompanying coordinate tuples are intersection points. Although the qcs-curves are drawn as straight lines, they can have any shape as long as they are monotonic.

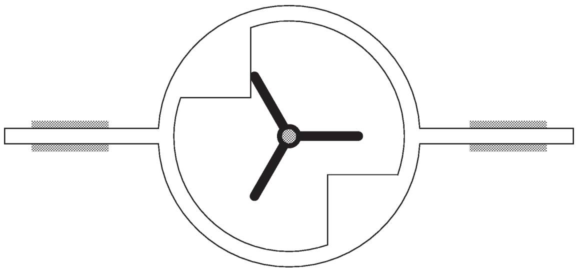


Fig. 38. The yoke and rotor device.

The constraints thus define an entire family of solutions a designer can explore in order to adapt an initial conceptual design to meet the design requirements.

We illustrate this with a new example concerning the design of the yoke and rotor device shown in Fig. 38. Continuous counter-clockwise rotation of the rotor causes the yoke to oscillate left and right with a brief dwell between each change in direction.

We describe the device to SKETCHIT with the stylized sketch in Fig. 39(a). The desired behavior is to have each of the rotor blades engage each of the yoke faces in turn as shown in Fig. 39(b). From this input SKETCHIT generates the BEP-Model in Fig. 40.

The designer now has available a large family of designs specified by the BEP-Model and can at this point begin to specify additional design requirements.

Imagine that one requirement is that all strokes have the same length. A simple way to achieve this is to add additional constraints to the BEP-Model to constrain the yoke and

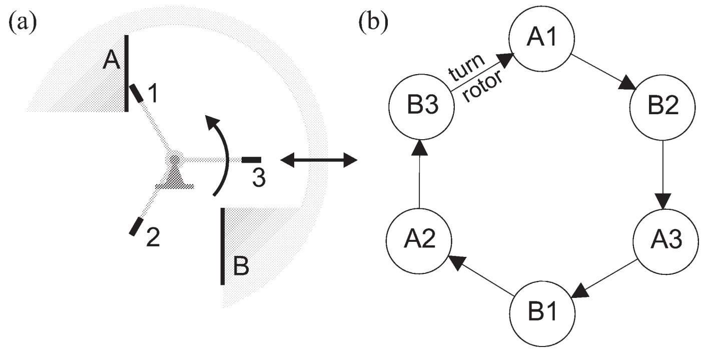
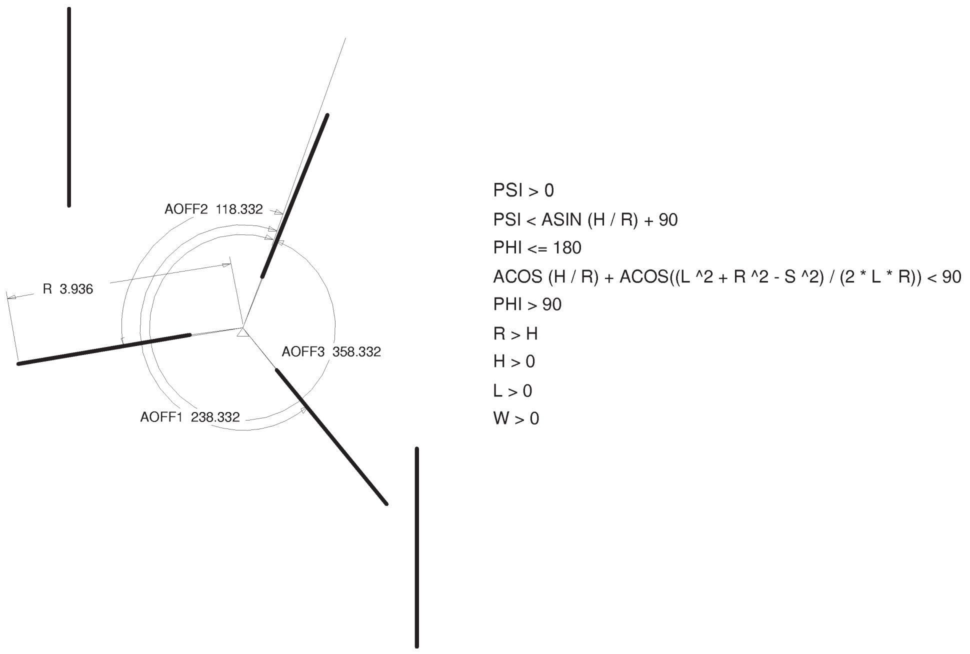


Fig. 39. (a) The stylized sketch of the yoke and rotor device. (b) The desired behavior. The letter and number in each node indicate which yoke face and which rotor face are engaged in that state. Turning the rotor is the external input that causes each of the transitions.



```
PSI > 0
PSI < ASIN (H / R) + 90
PHI <= 180
ACOS (H / R) + ACOS((L ^2 + R ^2 - S ^2) / (2 * L * R)) < 90
PHI > 90
R > H
H > 0
L > 0
W > 0
```

Fig. 40. A BEP-Model for the yoke and rotor; a representative sample of the parameters and constraints are shown. For simplicity, new variable names have been substituted for sets of variables constrained to be equal. For example, because all three rotor blades are constrained to have equal length, R replaces $R1$, $R2$, and $R3$.

rotor to be symmetric. For example, we constrain the rotor to be symmetric by constraining the rotor blades to be of equal length and to have equal spacing:

$$R1 = R2 = R3,$$

$$\mathrm{AOFF1} - \mathrm{AOFF2} = 120^\circ,$$

$$\text{AOFF3} - \text{AOFF1} = 120^\circ.$$

Imagine further that all strokes are required to be 1.0 cm long. We achieve this by adding the additional constraint: [23]

$$\text{LM29} - \text{LM27} = 1.0.$$

Finally, imagine that the dwell is required to be 40°, i.e., between each stroke, the rotor turns 40° while the yoke remains stationary. We can achieve this by adding one additional constraint: [24]

$$\text{LMG} - \text{LM8} = 40^\circ.$$

We can now invoke DesignView to find a solution to this augmented set of constraints; the solution will be guaranteed to produce both the basic desired behavior and the desired performance characteristics. We have been able to accomplish this design refinement simply by adding additional constraints to the BEP-Model.

10. Related work

Our task is to use a description of desired behavior to turn a sketch of a mechanical device into multiple families of working geometries. No previous work has directly addressed this particular task. The work that comes closest to ours is the work in design automation (specifically the approaches based on bond graphs and kinematic building blocks), the work in shape design, and the work in sketch understanding. The first three subsections of this section describe these three areas of research.

The remaining subsections describe supporting work, such as the work in qualitative physics and simulation, and work that indirectly touches on some of the issues involved in our task, such as the work in component connection models and geometric features.

10.1. Design automation

10.1.1. Bond graph approaches

Our techniques can be viewed as a natural complement to the bond graph techniques of the sort developed by Ulrich [51]. (See [28] for a comprehensive discussion of bond graphs.) Our techniques are useful for computing geometry that provides a specified behavior, but because of the inertia-free assumption employed by our simulator, our techniques are effectively blind to energy flow. Bond graph techniques, on the other hand, explicitly represent energy flow but are incapable of representing geometry.

[23] LM29 and LM27 are variables that SKETCHIT assigns to the extreme positions of the yoke. We obtain the names of these variables by using a graphical browser to inspect SKETCHIT's simulation of the device. Because we have constrained the yoke and the rotor to be symmetric, all strokes have the same length.

[24] LMG and LM8 are the variables that SKETCHIT assigns to the position of the rotor at the beginning and ending of one the dwells. Because we have constrained the yoke and the rotor to be symmetric, all six dwells have the same duration.

Prabhu and Taylor [37] and Welch and Dixon [52] extend the bond graph design approach to allow specification of positions and orientations of components, but they still do not design the shapes of interacting parts.

Their techniques synthesize a design using an abstract representation of behavior (bond graphs), then use library lookup to map to implementation. We use a similar paradigm, however, because our library contains interacting faces, while theirs includes complete components, we can design interacting geometry, while they cannot. Like our techniques, their techniques produce design variants.

10.1.2. Kinematic building blocks

Our techniques focus on the geometry of devices that have time varying engagements (i.e., variable kinematic topology). Therefore, our techniques are complementary to the well known design techniques for fixed topology mechanisms, such as the gear train and linkage design techniques in Erdman and Sandor [9].

Although these well known techniques are not applicable to our problem, they can be used to construct mechanism design tools for fixed topology devices as Rosen et al. [38] demonstrate. They describe a knowledge-based tool capable of designing dwell mechanisms based on cams, gears, and linkages.

There has been a lot of interest recently in automating the design of fixed topology devices. A common task is the synthesis of a device that transforms a specified input motion into a specified output motion: Kota and Chiou [29] use a matrix to represent the desired motion transformation, then use matrix decomposition to decompose this into basic building blocks. Subramanian and Wang [50] use iteratively deepening search to compose a sequence of “abstract mechanisms” that achieves the desired motion transformation. The bond graph based design techniques described above are also applicable to this task. All of these approaches are capable of producing design variants, but again, these techniques are not suitable for designing variable topology devices.

10.2. Shape design

Although our work crosses many research boundaries, it is most closely related to the work in shape design.

Joskowicz and Addanki [24] describe an automated tool for shape design. They start with two interacting shapes (2D profiles) and the desired behavior described as a c-space. They then modify the part shapes by adding and deleting line segments and arcs until the shapes produce the desired c-space. While they take the desired c-space as input, we compute a qc-space that will provide the desired behavior. Their techniques produce a single design instance, rather than a family of designs as SKETCHIT does.

Joskowicz [23] uses a set of local and global operators to simplify and abstract the c-space for a device in order to reduce irrelevant details. For example, one of the operators first divides the c-space boundaries (cs-curves) into monotonic pieces and then replaces those pieces with straight line segments. Another operator eliminates parts of the c-space that cannot be reached because of a particular choice of external inputs to the device. The closest application of these operators to our task is comparing the behavior of two devices: if their simplified and abstracted c-spaces are the same, the two devices provide the same qualitative behavior. Our task is still recognizably different: we want to generate designs that all produce the same qualitative behavior rather than recognizing designs that provide the same behavior.

Caine [2] and Joskowicz and Sacks [26] describe interactive design tools that allow the designer to modify shape by modifying c-space or vice versa. While we use library lookup to map from qc-space to geometry, they use numerical techniques, which can be computationally expensive, to map changes in c-space to changes in geometry. Their techniques produce a single design instance, while our techniques produce multiple families of designs. Finally, their techniques are interactive while ours are automatic.

There is some recent work in exploring the mapping between shape and behavior. Joskowicz et al. [27] use kinematic tolerance space (an extension of c-space) to examine how variations in the shapes of parts affect their kinematic behavior. Their task is to determine how a specified variation in shape affects behavior, ours is to determine what constraints on shape are sufficient to ensure the desired behavior.

Faltings [13] examines how much a single geometric parameter can change, when all others are held constant, without changing the place vocabulary (topology of c-space). Their task is to determine how much a given parameter can change without altering the current behavior, while ours is to determine the constraints on all the parameters sufficient to obtain a desired behavior.

Our task is most similar to that of Faltings and Sun [15]. They describe an interactive design system that modifies a user selected geometric parameter until there is a change in the place vocabulary, and hence a change in behavior. Their system then uses qualitative simulation to determine if the new behavior of the modified geometry matches a specified desired behavior. (They have a language for specifying desired behavior just as we do.) They use the techniques in [13] to determine how much a parameter must change in order to change the place vocabulary.

They modify c-space by modifying geometry, we modify qc-space directly. Because there are many changes in geometry that map to the same change in c-space, their search space is larger than ours. Also, our tool is automatic while theirs is interactive, and we can generate design variants while they cannot.

Gupta and Jakiela [18] describe a novel technique by which a known component "carves out" the shape of an unknown mating component. They require that one of the interacting shapes is known, but we do not. Also, they require a complete description of the desired motion of each component, while we do not.

10.3. Sketch understanding

There is little previous work in sketch understanding.[25] Narayanan et al. [34] use a diagram of a device to reason about its behavior, but they use a pre-parsed description of the behaviors of each component while we reason directly from the geometry of the interacting faces.

Faltings [12] suggests that a sketch is not a single qualitative model but instead represents a family of precise models. He demonstrates that, taking a sketch as a qualitative

[25] See [49] for a discussion of common sketching techniques used in engineering.

metric diagram (i.e., a line drawing with approximate dimensions), it is possible to compute what he calls kinematic topology. Kinematic topology [14] characterizes the topology of the free space regions of the device's c-space. Kinematic topology is an abstraction of place vocabulary [11] and as such, it often contains ambiguities. These ambiguities suggest behaviors that might possibly be obtained by modifying the geometry of the device. However, methods for determining which modifications will yield these other behaviors is still an open issue.

10.4. Qualitative physics

Our work builds upon the large and growing body of research in the field of qualitative physics. Weld [53] provides a comprehensive overview of the field. Here, to provide background for our qualitative simulation techniques, we discuss a representative sampling of the work in qualitative physics.

De Kleer [5] describes a program that produces causal explanations of the small signal behavior of electric circuits. The program computes behavior by using constraint propagation techniques to propagate the circuit's inputs through the circuit. These techniques are related to the work of Stallman and Sussman [48]. We use a simplified version of these propagation techniques to compute the motion of the bodies in a device.

De Kleer [6] describes a qualitative simulator based on confluences, i.e., qualitative differential equations. The behavior of each component in de Kleer's world is characterized by a set of qualitative states (operating regions); the behavior in each qualitative state is described by a set of confluences. The simulator computes all possible consistent qualitative states of the components in a device, and all possible transitions from one set of consistent qualitative states to another. Hence, the simulator computes an envisionment just as ours does.

De Kleer's simulator must be provided with an enumeration of the possible qualitative states (operating regions) of a component. In our domain, however, it is not possible to enumerate the possible qualitative states of a component because a component's behavior depends on the shapes of the other components with which it interacts.

Williams [54] describes a simulator that can reason about both the small signal and the large signal behavior of an electric circuit. The simulator computes the small signal model that applies in a particular operating mode, then predicts which parameters change, possibly causing a transition to another operating mode. The simulator then uses constraint analysis to determine which of the possible transitions actually happens first. (Kuipers [31] provides the mathematical underpinnings for these kinds of techniques.) This is very similar to the way our simulator computes the next event: it computes the possible events in each individual qcs-plane, then uses constraint propagation to determine which of these events can happen first.

Forbus [17] views the world from a process centered perspective rather than a device centered perspective. These techniques are suited to modeling processes like boiling which do not involve a fixed collection of "stuff": as the boiling process evolves, water turns to steam and leaves the system. However, the behavior of the kinds of mechanical devices we are interest in cannot be conveniently decomposed into a set of processes.

10.5. Simulation

SKETCHIT uses dynamic simulation to compute the behavior of a candidate c-space. Conventional dynamic simulators (e.g., [36], [19], and [47]) predict motion by numerically integrating the equations of motion of the device. Because we use a qualitative representation (qc-space), we cannot use numerical integration, and hence must use a qualitative simulator.

Faltings [11] describes a qualitative simulator for fixed-axis devices. The simulator is based on a representation called place vocabulary, which is a qualitative version of c-space. Place vocabulary decomposes the free space regions of a cs-plane into "places", regions in which the contact between the pair of components is uniform. There are three kinds of places: two-dimensional regions with no contacts, segments of cs-curves [26] (one contact exists), and intersections between cs-curves (two contact points exists). Place vocabulary also encodes the allowed transitions between the places. Because this representation is qualitative, ambiguities may arise as to which place transitions will actually occur in response to the applied inputs. In this case, their simulator computes all possible transitions.

Faltings' simulator is intended to be a kinematic simulator, and hence has a limited ability to reason about forces. [27] Our simulator, on the other hand, is a dynamic simulator, i.e., a simulator that computes the motion resulting from applied forces.

Forbus et al. [16] extend Faltings' techniques to produce a dynamic simulator. Their simulator models inertia, while we assume that motion is inertia-free. They represent forces as qualitative vectors, we do not. Our representation for forces is designed to eliminate the ambiguity that occurs when summing qualitative force vectors, thereby reducing branching of the simulation.

10.6. Computing numerical C-space

We obtain the first candidate qc-space by abstracting the numerical c-space of the sketch. Because the abstraction process requires only a partial description of the numerical c-space of the sketch, we developed simplified, special purpose techniques that compute just the required information. There are many general purpose techniques for computing the complete numerical c-space of a device. Lozano-Pérez [32], for example, describes an algorithm for computing the configuration space of two polygons that translate in the plane without rotation. Brost [1] and Caine [2] describe algorithms for the case in which the polygons rotate as well as translate. More suited to our needs is the work of Joskowicz and Sacks [25]. They compute the full c-space for a fixed-axis device by computing the cs-plane for each pair of interacting parts in the device.

10.7. Component-connection models

A common abstraction used in qualitative reasoning about physical systems is the component-connection model: each component has a set of ports; each port is associated

[26] They use the term constraint curve rather than cs-curve.

[27] Kinematics is the study of motion without reference to the forces that cause that motion. Kinematic simulation is commonly used to analyze the motion of devices that have a motion source applied to each degree of freedom.

with a parameter such as voltage or fluid flow rate. A set of constraints characterizes the relationships between the parameters of a particular component's ports. Components are connected at their ports; when two ports are connected, they share a common parameter. All of the interesting behavior occurs inside the components; the connections simply propagate parameter values.

By contrast, for the devices we are interested in, all of the interesting behavior occurs at the interaction between two components (i.e., the interaction between a pair of faces). Although the component-connection techniques do not apply directly to the problems in SKETCHIT's domain, they do address some relevant issues. Therefore, this section describes several examples of work in the area.

Doyle's [8] task is to hypothesize a structure that achieves a set of observable events; ours is to find geometry that achieves a desired behavior. He constructs hypotheses by connecting together primitive mechanisms (components). Each primitive mechanisms has a quantity type associated with its cause (input) and effect (output). Two primitive mechanisms can be connected together only if they have compatible quantity types. This serves as a primary source of constraint for limiting the generation of hypotheses. In our work, the sketch is the primary source of constraint for limiting search: we consider only devices that are similar to the initial sketch.

Falkenhainer and Forbus [10] describe a program that uses a library of model fragments and a description of the structure of a device to construct a model suitable for answering a user query about the device. The goal is to find the simplest model adequate to answer the query. Each of the model fragments describes one possible behavior of a component (i.e., one possible set of constraints relating the parameters at the ports). The program's task is to determine which of the possible behaviors actually occurs in the context of the overall device.

Nayak [35] describes a similar system. His task is to construct a model that provides a causal explanation of a device's behavior. The expected behavior of the device provides constraint for limiting the search for an adequate model. The expected behavior is described as a desired causal path, for example, "how does the temperature of the thermistor determine the angular deflection of the pointer?"

Mashburn and Anderson [33] extend the methods of Falkenhainer and Forbus to produce a system that guarantees that the model is complete, i.e., that there are enough equations to solve for the desired quantity.

Davis [4] describes a system that performs circuit diagnosis. His task is to determine which components, if malfunctioning, could account for the discrepancy between the observed behavior of a device and the correct behavior. Said differently, the task is to find a model that predicts the observed behavior rather than the correct (intended) behavior.

The primary task of each of these systems is to determine what role each part of the device plays in achieving the overall behavior. During the reverse engineering and generalization process, SKETCHIT has a similar task of determining what role each part of the device plays in achieving the *desired* overall behavior (see Fig. 21). Hence, the techniques used in these systems could be used to extend SKETCHIT's reverse engineering/re-synthesis paradigm to the component-connection domain.

10.8. Reverse engineering

Before SKETCHIT can synthesize new designs for a device, it must reverse engineer the original sketch. Shrobe [42] has also examined the task of reverse engineering, but his work is in the domain of linkages. He numerically simulates the kinematics of the linkages using Kramer's TLA [30], then parses the simulation to identify a set of common behaviors such as dwell and frequency doubling. These techniques might be used to extend SKETCHIT's reverse engineering/re-synthesis paradigm to the linkage domain.

10.9. Geometric features

In SKETCHIT's domain, the behavior of a device is determined by how the component shapes interact with each other. However, there are other domains in which the geometric features of individual components determine important design properties. Here we consider some of those domains.

For example, Hirschtick [20] describes a knowledge based tool that assists in the design of aluminum extrusions. The tool is rule-based and works in the domain of extrusions (cross-sections) built from line segments and arcs. The rules trigger off of patterns of geometric features. One rule, for example, states that a thick wall and a thin wall that meet at a corner should have a fillet. The program begins by identifying the important features of the extrusion: walls, corners, hollow cavities, fillets, etc., then applies the rule-base to produce manufacturing advice.

Dixon et al. [7] describe a similar manufacturing advisor. However, they use a feature based geometric design tool to construct a geometric model of the device, rather than recognizing features in a conventional CAD model of the device. Their domain is aluminum extrusions, castings, and injection molded plastic.

Wolter and Chandrasekaran [55] describe a feature-oriented design system capable of representing a wide range of functions. They represent geometry with a hierarchy of structures called geomes. A geome is a collection of geometric elements with constraints on how those elements are combined. They label a geome with the function for which it is commonly used for. For example, a geome consisting of a cylindrical rod inside a round hole of the same diameter would be labeled a pivot. The designer constructs a design by assembling geomes that provide the desired functions.

They also provide abstract geomes, geomes that have no implementation, allowing the designer to describe the intended function of a device rather than the structure. The revolute-constraint geome, for example, is a pivot with no implementation. The designer can later refine the abstract geomes to more specific geomes that do have geometric implementations.

Their functional language can represent static features (e.g., a slot) or constant contact (e.g., a pin in a hole or a rack and pinion). They cannot handle functions that require intermittent contact such as the function of the lever and hook in the circuit breaker.

10.10. Representing function

There is a wealth of research in representing and reasoning about function. Hodges [21], for example, describes a representation for capturing the behavior and function of the parts of mechanical devices. This representation is used to determine when a device may be useful for a task for which it was not originally intended. Iwasaki et al. [22] also describe a language for representing the function of the parts of a device. Their goal is to explicitly record how the device is supposed to work in addition to recording the device's structure, thereby enabling services like automated design verification.

During reverse engineering, SKETCHIT determines what role each part of the device plays in achieving the desired overall behavior. The program is in effect computing a description of the function of each part. Hence, our work is complementary to approaches that reason from a representation of function in order to perform other useful tasks.

10.11. Algebraic constraints

SKETCHIT produces output in the form of a BEP-Model, a parametric model with constraints that ensure the desired behavior. Serrano and Gossard [40] describe a system called MATHPAK that is suitable for solving the constraints found in a BEP-Model. [28] Similarly, Serrano [41] describes a system for efficiently solving systems of algebraic constraints like those contained in a BEP-Model.

11. Future work

SKETCHIT can currently repair a limited range of flaws in the original sketch. As Section 4 described, we are continuing to develop techniques for repairing more serious kinds of flaws.

We currently use state transition diagrams to specify the desired behavior of a device in terms of a desired sequence of engagements. We would like to develop a behavior specification language that is more like the verbal language that engineers use to describe the behavior of devices. For example, we would like to specify the desired behavior using common engineering terms like "ratchet", "clutch", or "trip mechanism". We believe that the state transition diagram language is a good substrate upon which to implement this better language. For example, many engineering terms, such as "ratchet", have a direct translation into a desired sequence of engagements, and hence are simply a macro on top of the state transition diagram language.

We are also working to expand the class of devices that SKETCHIT can handle. Currently, our techniques are restricted to fixed-axis devices. Although this constitutes a significant portion of the variable topology devices used in actual practice (see [39]), we would like extend our techniques to handle particular kinds of non-fixed-axis devices.

We are currently exploring a commonly occurring class of devices in which a pair of parts has three degrees of freedom (rather than two) but the qc-space is still tractable. These devices have switchable degrees of freedom: for each different mode of operation one of the degrees of freedom is switched off so that, at any given time, at most two degrees of freedom are active. SKETCHIT could represent these kinds of devices with a set

[28] DesignView, the system we use to solve the constraints of a BEP-Model, is based on MATHPAK.

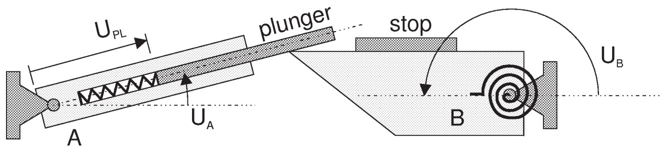


Fig. 41. A device with switchable degrees of freedom. The device employs two rotors, labeled A and B. Rotor A has a spring-loaded plunger protruding from it, B has a spring pushing it toward a stop.

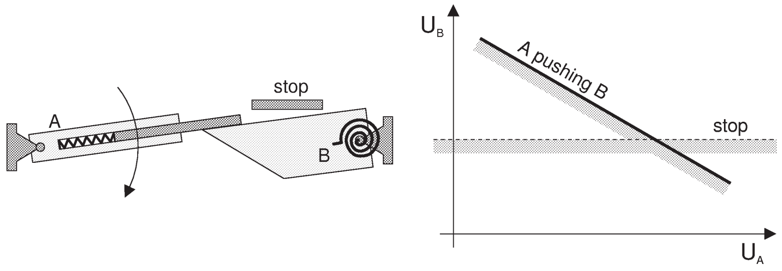


Fig. 42. The first mode of operation and the corresponding qc-space. The plunger's degree of freedom is switched off.

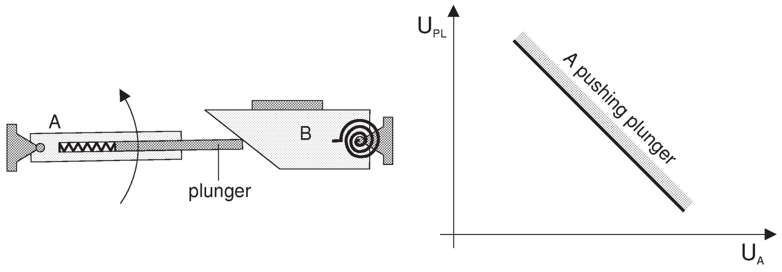


Fig. 43. The second mode of operation and the corresponding qc-space. Rotor B's degree of freedom is switched off.

of qcs-planes, one for each mode of operation. The simulator would select the appropriate qcs-plane for each step of simulation.

Fig. 41 shows an example consisting of two rotors, labeled A and B. Rotor A has a spring-loaded plunger protruding from it, B has a spring pushing it toward a stop. Devices similar to this are used, for example, to index the film advance in cameras. In the first mode of operation the plunger has no degrees of freedom. As rotor A turns clockwise it pushes rotor B as shown in Fig. 42. Eventually the rotors disengage and B's spring pushes B against its stop. In the second mode of operation, rotor B has no degrees of freedom while the plunger has one: rotor A turns counterclockwise causing the plunger to be depressed by engaging rotor B as shown in Fig. 43. In normal use, the device would alternate between these two modes.

Fig. 42 shows the qc-space for the first mode. The qc-space is a plane defined by the degrees of freedom of the rotors. Because the plunger has zero degrees of freedom in this mode, it does not appear in the qc-space. Fig. 43 shows the qc-space for the second mode. This qc-space is also a plane, but this time it is defined by the degrees of freedom of rotor A and the plunger. At any point in the simulation, one or the other of these qcs-planes will be active. The simulator's new task is to select the applicable qcs-plane for each step of the simulation.

We would also like to relax the restriction to frictionless engagements. One simple approach would be to model friction as a force that either dominates the force balance or is negligible. Using this model the simulator would have an additional property to branch on when computing force balances: whether or not the friction from an engagement dominates the other forces applied to a body. Once SKETCHIT determines which engagements should provide high friction and which should not, it must add appropriate constraints to the BEP-Model.

These constraints can be expressed in terms of the friction cone, which defines the range of forces (i.e., the range of orientations of a force) that a surface can resist without slipping. The size of the friction cone is a function of the coefficient of friction and the geometric parameters of the interacting faces. To ensure that an engagement provides high friction, SKETCHIT must add constraints to the BEP-Model that ensure the engagement forces are inside the friction cone. Conversely, to ensure that the friction is negligible, the program must add constraints that ensure the engagement forces are outside the friction cone.

After making these extensions to our system, the next task is to determine how well these techniques scale to design problems more complex than the two working examples reported here. [29]

We are beginning to explore how our techniques can be applied to other problem domains. For example, we believe that the BEP-Model will be useful for kinematic tolerance analysis (see [3] for an overview of tolerancing). Here the task is to determine if a given set of variations in the shapes and locations of the parts of a device will compromise the desired behavior. A possible approach to this task is to determine if the variations are contained in the family of working designs defined by the BEP-Model. For example, a simplistic implementation would use Monte Carlo simulation to determine if a large number of designs randomly selected from the specified set all satisfy the constraints of the BEP-Model.

We have also begun to explore design rationale capture. We believe that the constraints of the BEP-Model will be a useful form of design documentation, serving as a link between the geometry and the desired behavior. The constraints might, for example, be used to prevent subsequent redesign efforts from modifying the geometry in a way that compromises hard won design features in the original design.

12. Conclusion

We have demonstrated that SKETCHIT can generate multiple families of designs from a single sketch and that it can repair a limited range of flaws in the initial design.

[29] We have tested the program on three examples: the circuit breaker and yoke and rotor examples described here and a firing mechanism from a single action revolver.

SKETCHIT represents each of the new families with a BEP-Model, parametric geometry with constraints that ensure the geometry produces the desired behavior. SKETCHIT creates different families by changing the motion types of the components and by selecting different geometries for the engagement faces. As our examples illustrate, the new designs that SKETCHIT produces include a wide range of new design alternatives.

SKETCHIT is able to perform these tasks because of qc-space, a new representation for mechanical behavior. Qc-space captures the behavior of a device but abstracts away its particular implementation, thus providing the opportunity to select new implementations.

Given the intimate connection between shape and behavior, design of mechanical artifacts is typically conceived of as the modification of shape to achieve behavior. But if changes in shape are attempts to change behavior, and if the mapping between shape and behavior is quite complex [2], then, we suggest, why not manipulate a representation of behavior? Our qualitative c-space is just such a representation. We suggest that it is complete and yet offers a far smaller search space. It is complete because any change in shape will produce a c-space that maps to a new qc-space differing from the original by at most changes in relative locations and qualitative slopes of the qcs-curves. Qc-space is far smaller precisely because it is qualitative: often many changes to the geometry map to a single change in qc-space. Finally, it is an appropriate level of abstraction because it isolates the differences that matter: changes in the relative locations and qualitative slopes are changes in behavior.

One reason this work is important is that sketches are ubiquitous in design. They are a convenient and efficient way both to capture and communicate design information. By working directly from a sketch, SKETCHIT takes us one step closer to CAD tools that speak the engineer's natural language.

Appendix A. Deriving the library

In this appendix we describe the interaction library, providing a derivation of one of the library entries. We use as our example a library entry that implements qcs-curve H in Fig. 19, for the case in which q_1 is rotation and q_2 is translation (i.e., rotation in the negative direction causes translation in the positive direction). [30]

This implementation uses the two flat faces whose parameterization is the same as that in Fig. 26, except that x is measured positive to the right. Our goal is to derive a set of constraints on the parameters such the faces implement a monotonic qcs-curve with the same slope as qcs-curve H.

As Fig. 26 shows, there are 7 parameters that characterize the pair of faces (we do not count θ and x because they measure positions, i.e., they are position parameters not geometric parameters). To obtain constraints ensuring that the faces implement qcs-curve H, we must identify the regions in the 7-dimensional parameter space for which the faces produce a monotonic curve of correct slope. Finding all such regions, and

[30] We focus on curve type H, rather than types F or D which are used in the circuit breaker example, because the coordinate frames are more convenient. We do, however, show how to transform this implementation into implementations for curves of type F and D.

expressing the result as a set of algebraic constraints is a difficult, if not intractable, task. Instead, we look for individual regions for which the constraints are simple. The more regions we can identify, the more ways the program will have to implement a design.

To guide us in the search for the simple regions, we take inspiration from the kinds of geometry that designers commonly use to achieve a particular kind of interaction. For example, a common way to implement curve H using the geometry of Fig. 26 is if the geometry acts like a cam and offset follower: a design for which the translating face does not pass through the pivot of the rotating face. With this as our inspiration, our goal is to find a set of constraints which describe the class of solutions corresponding to a cam with offset follower.

Our first constraint is that the two faces exist, that is, they have non-zero length. Defining w as the length of the translating face and L as the length of the rotating face, we trivially obtain:

$$w > 0 \tag{A.1}$$

$$L > 0 \tag{A.2}$$

To ensure that the translating face does not pass through the pivot (a hallmark of cam-and-offset-follower behavior), we define the follower offset, h, subject to the constraint:

$$h > 0 \tag{A.3}$$

We define h such that when one looks down the positive x-axis, the translating face is to the left.

To ensure that the rotor can actually engage the slider, we must ensure that the rotor is long enough to reach the slider. To this end we construct the distance from the pivot to each end of the rotating face. The longer of the two distances we label r, the other we label s (because of the constraint expressed by Equation (A.7), one distance is always larger than the other). r and s are related to the length of the rotating face by:

$$r = \left(s^2 + L^2 - 2sL\cos(\phi)\right)^{1/2} \tag{A.4}$$

For the rotor to be long enough to engage the slider, r must satisfy:

$$r > h \tag{A.5}$$

If s is sufficiently large, when the rotor is vertical and pointing upward ($\theta \approx \pi/2$), the slider face will be able to pass under the rotor face. As a result, there will be two separate ranges of angle for which the rotor and slider will be able to engage, one range to the right of vertical and another to the left of vertical. In this case, the interaction between the rotor and slider will be two disjoint cs-curves, and thus this geometry will not be a valid implementation of curve H which is a single curve. To ensure that this kind of

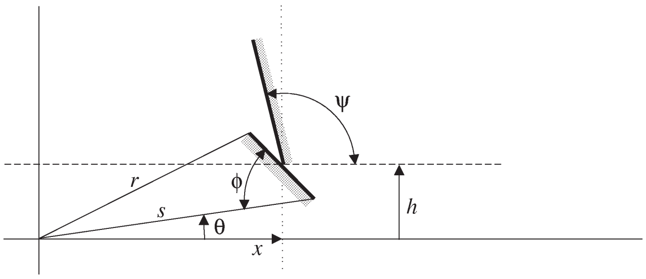


Fig. A.1. The leading edge of the rotor is shorter than the trailing edge. Turning the rotor counterclockwise pushes the slider to the right.

disjointed interaction does not occur, we must constrain length s.[31] An overly conservative constraint, which we use for its simplicity, is:

$$s < h \tag{A.6}$$

A less restrictive, but equally effective constraint is $s < h + w\cos(\psi - \pi/2)$. This constraint ensures that when the rotor is vertical ($\theta = \pi/2$) the top of the slider face is higher than the bottom of the rotor face. For historical reasons, the program uses the more conservative constraint.

Defining rotation positive counterclockwise, we can refer to the line labeled r in Fig. 26 as the leading edge of the rotor, and the line labeled s as the trailing edge.

If the leading edge is shorter that the trailing edge, the cs-curve will not have the desired slope.[32] Consider a configuration in which the bottom end of the slider face touches the middle of the rotor face and the rotor turns counterclockwise as shown in Fig. A.1. If the leading edge is shorter than the trailing edge, the radius from the pivot to the contact point will get longer. Because this radius gets longer, the rotor will push the slider to the right. In this case, the cs-curve will have (at least locally) a slope like qcs-curve B in Fig. 19, rather than like qcs-curve H. Hence, we must constrain the leading edge to be longer than the trailing edge. We express this constraint in terms of the angle ϕ between the trailing edge and the rotor face:

$$\phi > \pi/2 \tag{A.7}$$

Consider the configuration in which the tip of the rotor is touching the bottom of the slider face as shown in Fig. A.2. If the rotor face is horizontal in this configuration (it is not horizontal in the figure), then the pair of faces will act as a stop: the slider face will be able to slide freely along the rotor face, but the rotor will be prevented from rotating clockwise. The corresponding qcs-curve will locally be vertical, not diagonal like qcs-curve H. Hence,

[31] We can use the disjointed solution if we know that during the normal operation of the device, the device assumes configurations from only one range of engagement. Although for this implementation of curve H we can explicitly constrain against multiple ranges of engagement, this is not possible for our other implementation of curve H that uses geometry like that in Fig. 27. That implementation exhibits two ranges of engagement. In that case, it is necessary to place additional constraints on the design to ensure that only the desired range of engagement occurs during normal operation of device. These additional constraints are described in [43].

[32] We assume that $0 \leqslant \phi \leqslant \pi$.

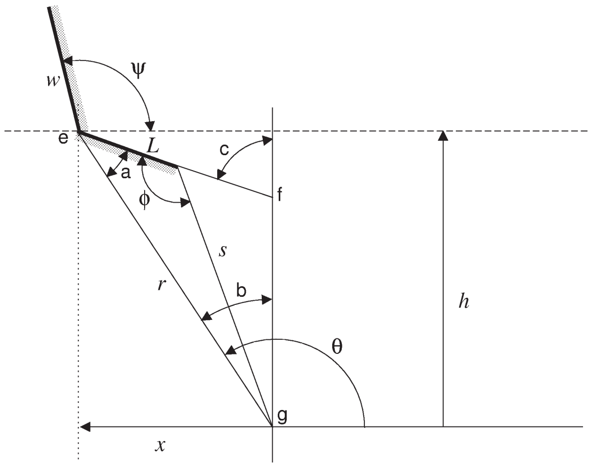


Fig. A.2. When in this configuration, the rotor face must not be horizontal.

we must constrain the rotor face so that it is not horizontal when in this configuration. If we define c as the angle between the rotor face and vertical (see Fig. A.2), the appropriate constraint is:

$$c < \pi/2$$

Our task now is to express c in terms of the parameters of the faces. Fig. A.2 shows the parameters as well as some intermediate variables we use in the derivation. We start by considering triangle e-f-g shown in the figure. Because a and b are two angles of this triangle, and c is the complement of the third angle:

$$c = a + b$$

Using the law of cosines we obtain for a:

$$a = \arccos\big((L^2 + r^2 - s^2)/2Lr\big)$$

By inspection of the figure we obtain for b:

$$b = \arccos(h/r)$$

Combining the previous four expressions, we obtain the desired constraint:

$$\arccos\big((L^2 + r^2 - s^2)/2Lr\big) + \arccos(h/r) < \pi/2 \qquad \text{(A.8)}$$

Our expression for a assumes that $0 \leqslant \phi \leqslant \pi$. Eq. (A.7) enforces the lower bound, but we require an explicit constraint for the upper bound:

$$\phi \leqslant \pi \qquad \text{(A.9)}$$

For the qcs-curve to be monotonic, the tip of the rotor must never move tangent to the slider face. This in turn requires that the angle between the slider face and the leading edge of the rotor must always be greater than $\pi/2$. If we label this angle z, as shown in Fig. A.3, the constraint is:

$$z > \pi/2$$

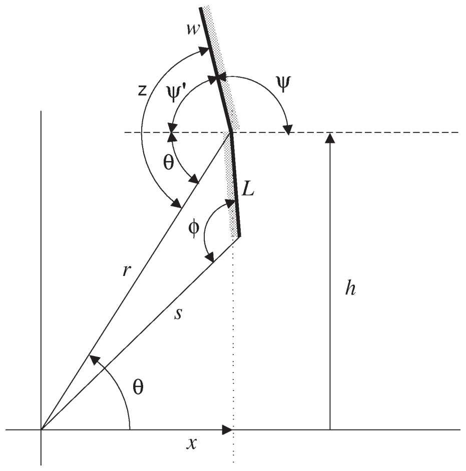


Fig. A.3. The rotor face must not move tangent to the slider face.

The worst case is when the rotor angle is at its smallest, which occurs when the tip of the rotor touches the lower end of the slider face as shown in Fig. A.3. If we define ψ' as the complement of the angle of the slider face, then by inspection of the figure:

$$z = \psi' + \theta$$

Because ψ' is the complement of ψ we can write:

$$\psi' = \pi - \psi$$

For the tip of the rotor to touch the bottom of the slider face, θ's value must be:

$$\theta = \arcsin(h/r)$$

Combining the previous four expressions we obtain the final form of the constraint:

$$\psi < \arcsin(h/r) + \pi/2 \tag{A.10}$$

If the slider face angle ψ is equal to 0, only one end of the slider face will touch the rotor. If ψ is less than 0, the rotor can touch the back side of the slider face, but this should not happen because contact is allowed only on the outside surface of a part. To ensure that the rotor cannot touch the back side of the rotor face, and that the contact can actually be face contact (rather than contact at just one end of the face) we enforce the constraint:

$$\psi > 0 \tag{A.11}$$

Eq. (A.1) through Eq. (A.11) are the complete set of constraints sufficient to ensure that the geometry in Fig. 26 implements qcs-curve H. Now to complete this library entry, we must derive expressions for the end point coordinates of the qcs-curve. One end point corresponds to the configuration in Fig. A.3. This end point is the upper left end point of curve H. Its coordinates are:

$$\theta_1 = \arcsin(h/r) \tag{A.12}$$

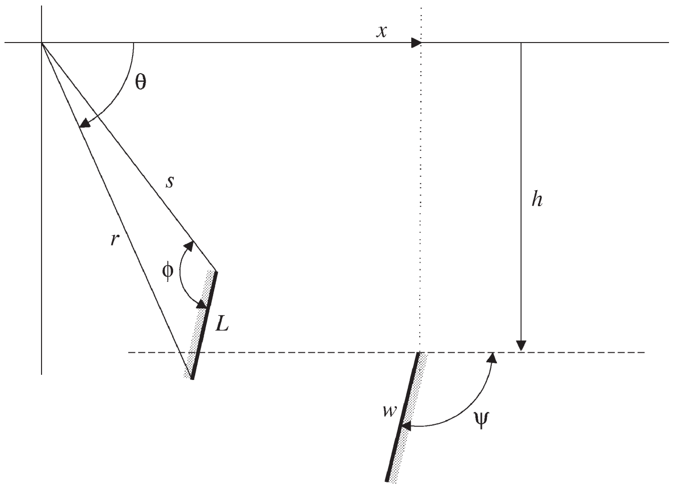


Fig. A.4. The parameterization of a pair of faces used to implement qcs-curve B. The rotating face rotates about the origin, its position measured positive *counterclockwise* with angle θ. The translating face translates horizontally, its position measured positive to the right with x. Shading indicates the back sides of the faces, i.e., the sides that are inside the solids containing the faces.

$$x_1 = r\cos(\theta_1) \tag{A.13}$$

The second end point corresponds to the configuration in Fig. A.2. Its coordinates are (note that $x_2 = -x_1$):

$$\theta_2 = \pi - \arcsin(h/r) \tag{A.14}$$

$$x_2 = -r\cos(\theta_1) \tag{A.15}$$

By appropriate use of coordinate transformations, we can use this pair of faces to implement several other qcs-curves.[33] For example, if we measure the position of the slider positive to the left rather than to the right, we have an implementation for curve F. If we flip the geometry over as in Fig. A.4, still measuring the rotor angle positive counterclockwise, we have an implementation for curve B. If we flip the geometry over and measure the slider position positive to the left rather than to the right, we have an implementation for curve D. Thus, with the same basic geometry and constraints, we can implement all four kinds of diagonal curves for the case in which q_1 is rotation and q_2 is translation. By the obvious coordinate transformation, we can also use this geometry to implement all four kinds of diagonal curves for the case in which q_1 is translation and q_2 is rotation.

[33] The coordinate transformations change (there are sign changes) the expressions for the coordinates of cs-curve end points (e.g., Eq. (A.12)) but they do not affect the constraints that ensure the curve is monotonic with proper slope (e.g., Eq. (A.1)).

References

[1] R.C. Brost, Analysis and planning of planar manipulation tasks, Technical Report CMU-CS-91-149, Carnegie Melon University, Pittsburgh , PA, 1991.
[2] M.E. Caine, The design of shape from motion constraints, Technical Report 1425, MIT AI Lab., Cambridge, MA, September 1993.
[3] K.W. Chase, A.R. Parkinson, A survey of research in the application of tolerance analysis to the design of mechanical assemblies, Research in Engineering Design 3 (1991) 23–37.
[4] R. Davis, Diagnostic reasoning based on structure and behavior, Artificial Intelligence 24 (1984) 347–410.
[5] J. de Kleer, Causal and teleological reasoning in circuit recognition, Ph.D. thesis, Massachusetts Institute of Technology, Cambridge, MA, September 1979.
[6] J. de Kleer, J.S. Brown, A qualitative physics based on confluences, Artificial Intelligence 24 (1984) 7–83.
[7] J.R. Dixon, E.C. Libardi Jr., S.C. Luby, M. Vaghul, Expert systems for mechanical design: examples of symbolic representations of design geometries, Engineering with Computers 2 (1) (1987) 1–10.
[8] R.J. Doyle, Hypothesizing device mechanisms: opening up the black box, Technical Report 1047, MIT AI Lab., Cambridge, MA, 1988.
[9] A.G. Erdman, G.N. Sandor, Mechanism Design: Analysis and Synthesis, Vol. 1, Prentice-Hall, Englewood Cliffs, NJ, 1984.
[10] B. Falkenhainer, K.D. Forbus, Compositional modeling: finding the right model for the job, Artificial Intelligence 51 (1991) 95–143.
[11] B. Faltings, Qualitative kinematics in mechanisms, Artificial Intelligence 44 (1990) 89–119.
[12] B. Faltings, Qualitative models in conceptual design: a case study, in: Reasoning with Diagrammatic Representations, Papers from the 1992 Spring Symposium, Technical Report SS-92-02, AAAI Press, 1992, pp. 69–74.
[13] B. Faltings, A symbolic approach to qualitative kinematics, Artificial Intelligence 56 (1992) 139–170.
[14] B. Faltings, E. Baechler, J. Primus, Reasoning about kinematic topology, in: Proc. IJCAI-89, Detroit, MI, 1989, pp. 1331–1336.
[15] B. Faltings, K. Sun, Computer-aided creative mechanism design, in: Proc. IJCAI-93, Chambery, France, 1993, pp. 1451–1457.
[16] K.D. Forbus, P. Nielsen, B. Faltings, Qualitative spatial reasoning: the clock project, Technical Report 9, Northwestern University, The Institute for the Learning Sciences, Evanston, IL, 1991.
[17] K.D. Forbus, Qualitative process theory, Artificial Intelligence 24 (1984) 85–168.
[18] R. Gupta, M.J. Jakiela, Simulation and shape synthesis of kinematic pairs via small-scale interference detection, Research in Engineering Design 6 (1994) 103–123.
[19] E.J. Haug, Computer-Aided Kinematics and Dynamics of Mechanical Systems, Volume I: Basic Methods, Allyn and Bacon, Boston, MA, 1989.
[20] J.K. Hirschtick, Geometric feature extraction using production rules. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, 1986.
[21] J. Hodges, Naive mechanics, a computational model of device use and function in design improvisation, IEEE Expert 7 (1) (1992) 14–27.
[22] Y. Iwasaki, R. Fikes, M. Vescovi, B. Chandrasekaran, How things are intended to work: capturing functional knowledge in device design, in: Proc. IJCAI-93, Chambery, France, 1993, pp. 1516–1522.
[23] L. Joskowicz, Simplification and abstraction in kinematic behaviors, in: Proc. IJCAI-89, Detroit, MI, 1989, pp. 1337–1342.
[24] L. Joskowicz, S. Addanki, From kinematics to shape: an approach to innovative design, in: Proc. AAAI-88, St. Paul, MN, 1988, pp. 347–352.
[25] L. Joskowicz, E. Sacks, Computational kinematics, Artificial Intelligence 51 (1991) 381–416.
[26] L. Joskowicz, E. Sacks, Configuration space computation for mechanism design, in: Proc. IEEE Robotics and Automation Conference, 1994.
[27] L. Joskowicz, E. Sacks, V. Srinivasan, Kinematic tolerance analysis, in: 3rd ACM Symposium on Solid Modeling and Applications, 1995.
[28] D. Karnopp, R. Rosenberg, System Dynamics: A Unified Approach, John Wiley, New York, 1975.
[29] S. Kota, S.-J. Chiou, Conceptual design of mechanisms based on computational synthesis and simulation of kinematic building blocks, Research in Engineering Design 4 (1992) 75–87.

[30] G.A. Kramer, Solving geometric constraint systems, in: Proc. AAAI-90, Boston, MA, 1990, pp. 708–714.
[31] B.J. Kuipers, Qualitative simulation, Artificial Intelligence 29 (1986) 289–388.
[32] T. Lozano-Pérez, Spatial planning: a configuration space approach, IEEE Transactions on Computers C-32 (2) (1983).
[33] T.A. Mashburn, D.C. Anderson, Automatically deriving behavior constraints for performance variables in mechanical design, Research in Engineering Design 6 (1994) 85–102.
[34] N.H. Narayanan, M. Suwa, H. Motoda, How things appear to work: predicting behaviors from device diagrams, in: Proc. AAAI-94, Seattle, WA, 1994, pp. 1161–1167.
[35] P.P. Nayak, L. Joskowicz, S. Addanki, Automated model selection using context-dependent behaviors, in: Proc. AAAI-92, San Jose, CA, 1992, pp. 710–716.
[36] P.E. Nikravesh, Computer-Aided Analysis of Mechanical Systems, Prentice-Hall, Englewood Cliffs, NJ, 1988.
[37] D.R. Prabhu, D.L. Taylor, Synthesis of systems form specifications containing orientations and positions associated with flow variables, in: Advances in Design Automation, Vol. 1, Computer-Aided and Computational Design, 1989, pp. 273–279.
[38] D. Rosen, D. Riley, A. Erdman, A knowledge based dwell mechanism assistant designer, Journal of Mechanical Design 113 (1991) 205–212.
[39] E. Sacks, L. Joskowicz, Automated modeling and kinematic simulation of mechanisms, Computer-Aided Design 25 (2) (1993) 106–118.
[40] D. Serrano, D.C. Gossard, Combining mathematical models with geometric models in CAE systems, in: Proc. 1986 ASME International Computers in Engineering Conference and Exhibition, 1986, pp. 277–284.
[41] D. Serrano, Constraint management in conceptual design, Ph.D. thesis, Massachusetts Institute of Technology, Cambridge, MA, 1987.
[42] H. Shrobe, Understanding linkages, in: Proc. AAAI-93, Chambery, France, 1993, pp. 620–625.
[43] T.F. Stahovich, SKETCHIT: a sketch interpretation tool for conceptual mechanical design, Technical Report 1573, MIT AI Lab., Cambridge, MA, March 1996.
[44] T.F. Stahovich, R. Davis, H. Shrobe, Turning sketches into working geometry. in: Proc. 7th International ASME Conference on Design Theory and Methodology, 1995.
[45] T.F. Stahovich, R. Davis, H. Shrobe, Generating multiple new designs from a sketch, in: Proc. AAAI-96, Portland, OR, 1996, pp. 1022–1029.
[46] T.F. Stahovich, R. Davis, H. Shrobe, Qualitative rigid body mechanics, in: Proc. AAAI-97, Providence, RI, 1997.
[47] T.F. Stahovich, Computational tools for conceptual design, Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, 1990.
[48] R.M. Stallman, G.J. Sussman, Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis, Technical Report Memo 380, MIT AI Lab., Cambridge, MA, September 1976.
[49] R.F. Steidel Jr., J.M. Henderson, The Graphic Languages of Engineering, John Wiley, New York, 1983.
[50] D. Subramanian, C.-S. (E.) Wang, Kinematic synthesis with configuration spaces, in: Proc. 7th International Workshop on Qualitative Reasoning about Physical Systems, May 1993, pp. 228–239.
[51] K.T. Ulrich, Computation and pre-parametric design, Technical Report 1043, MIT AI Lab., Cambridge, MA, 1988.
[52] R.V. Welch, J.R. Dixon, Guiding conceptual design through behavioral reasoning, Research in Engineering Design 6 (1994) 169–188.
[53] D.S. Weld, J. de Kleer (Eds.), Readings in Qualitative Reasoning about Physical Systems, Morgan Kaufmann, San Mateo, CA, 1990.
[54] B.C. Williams, Qualitative analysis of MOS circuits, Artificial Intelligence 24 (1–3) (1984) 281–346.
[55] J. Wolter, P. Chandrasekaran, A concept for a constraint-based representation of functional and geometric design knowledge, in: Proc. Symposium on Solid Modeling Foundations and CAD/CAM Applications, ACM Press, New York, 1991, pp. 409–418.