ELSEVIER

Artificial Intelligence 74 (1995) 1-53

Artificial Intelligence

Automated reasoning about machines *

Andrew Gelsey *

Computer Science Department, Hill Center for the Mathematical Sciences, Rutgers University, New Brunswick, NJ 08903, USA

Received June 1992; revised July 1993

Abstract

Numerical simulation is often used in predicting machine behavior, a basic capability for many tasks such as design and fault diagnosis. However, using simulators requires considerable human effort both to create behavioral models and to analyze and understand simulation results. I describe algorithms which automate the kinematic and dynamical analysis needed to create behavioral models and which automate the intelligent control of computational simulations needed to understand a machine's behavior over both short and long time scales. The input is a description of a machine's geometry and material properties, and the output is a behavioral model for the machine and a concise qualitative/quantitative prediction of the machine's long-term behavior. My algorithms have been implemented in a working program which can predict a machine's behavior over both short and long time periods. At present this work is limited to mechanical devices, particularly clockwork mechanisms.

1. Introduction

Predicting a machine's behavior is a basic capability for many tasks requiring reasoning about machines, such as diagnosis of malfunctioning machines, design of new machines, and redesign of existing machines. Numerical simulation is often used in predicting the behavior of machines and other physical systems. However, using simulation to predict machine behavior requires considerable human effort both to create the behavioral models on which the simulation is based and to analyze and understand the

* This research was supported by National Science Foundation grants IRI-8610241 and IRI-8812790 and by the Defense Advanced Research Projects Agency and the National Aeronautics and Space Administration under NASA grant NAG2-645.

* E-mail: gelsey@cs.rutgers.edu.

SSDI 0004-3702(94)00003-J

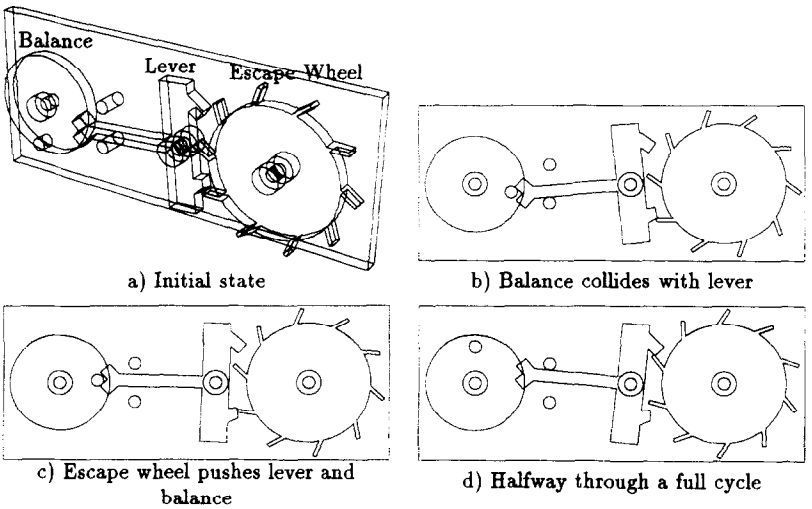


Fig. 1. Clock or watch escapement mechanism. (Note: hairspring attached to balance and mainspring attached to escape wheel are present in model but not shown in figures.)

results of the simulation. Automating this process requires basic artificial intelligence research in both spatial reasoning and reasoning about physical systems.

The research I describe in this article addresses two main problems:

- automated creation of behavioral models of machines directly from models of their raw physical structure;
- intelligent control of the computational experiments needed to reveal a machine's long-term behavior.

Together, the algorithms I present for solving these problems allow the automated prediction of a machine's behavior over both short and long time periods. At present this work is limited to mechanical devices, particularly clockwork mechanisms.

All the algorithms in this article have been implemented in a working program. The program's input is a description of the physical structure of a machine, which consists primarily of a precise numerical specification of the machine's geometry. From this input the program creates a behavioral model of the machine which it then uses to numerically simulate the machine's precise behavior when started from a number of intelligently chosen initial conditions. The program's final output is a concise qualitative/quantitative description of the machine's expected long-term behavior.

I will present an example which gives an overview of the entire process. The escapement mechanism in Fig. 1 keeps the average speed of a clock or watch constant by allowing the escape wheel, which is pushed clockwise by a strong spring, to advance by only one tooth for each oscillation of the balance. In Fig. 1(a) the motion of the escape wheel is blocked by the lever, and the balance is motionless and about to be driven counterclockwise by its attached spring. In Fig. 1(b) the balance has hit the lever. Impact force pushes the lever far enough to free the escape wheel, which then pushes both lever and balance as in Fig. 1(c). This pushing restores the energy the balance loses to friction, so that it can act as a harmonic oscillator in spite of damping. Finally,

in Fig. 1(d), the escape wheel and lever are locked together again, and the balance has been brought temporarily to a halt by its spring.

In this article, a behavioral model for a machine is considered to consist of two parts:

- the identification of a set of *state variables*: any particular state of the machine may be specified by assigning specific numerical values to each of the state variables;
- a description of how the state variables change with time, usually by differential equations.

The input to my program for the escapement example only specifies the shapes of the parts and their positions in space; this input does not indicate whether any of the parts are in contact or whether they impose any constraints on each other. Therefore, in order to identify a useful set of state variables, my program must use spatial reasoning to determine the kinematic properties of the escapement mechanism.

The program identifies three kinematic pairs, pairs of parts which mutually constrain each other's motion, by looking for pairs of parts having subparts with matching symmetries in corresponding positions. For example, in the escapement, the balance has a hole in it which has rotational symmetry about a certain axis, and the frame has a shaft having rotational symmetry about the same axis. Algorithms 2 and 3 in Section 2.1.1 specify the conditions under which matching symmetries result in a kinematic pair. For the escapement, my program determines that each of the three moving parts forms a revolute pair with the frame and thus is constrained only to rotate about a fixed axis. The program then partitions the moving parts into *kinematic subsystems* each having a single degree of freedom, using algorithms from Section 2.1.3. In this mechanism, none of the three moving parts are connected to each other by kinematic pairs, so there are three kinematic subsystems, each consisting of a single moving part. As a result, the program concludes that a reasonable set of state variables for the escapement consists of six variables: one position variable and one velocity variable for each of the three kinematic subsystems.

In order to generate the other half of the behavioral model for the escapement, the differential equations describing how the state variables change with time, the program must analyze the dynamics of the mechanism. My program can produce two separate behavioral models for a machine, based on distinctly different sets of approximations and simplifying assumptions. The user must choose between the two models: this article does not address model selection. However, the availability of at least two disparate models seems a necessary prerequisite for experimental investigation of how choices of approximations and simplifying assumptions influence the forms of behavioral models and the sorts of behavior the models predict. For the escapement mechanism, these models differ in their treatment of intermittent contacts between the moving parts, which are critical to the functioning of this mechanism. (Permanent contacts are handled with kinematic pairs, as described above.) The first model handles intermittent contacts between the moving parts of a mechanism by approximating the very small elastic distortions of the parts which give rise to contact forces. In the other model parts of the mechanism are treated as absolutely rigid bodies, and contact forces result from geometric constraints on the relative motions of parts in contact. Section 2.4 compares the two models. The other forces that must be modeled for the escapement mechanism are springs and friction, and my program uses simple linear models for both.

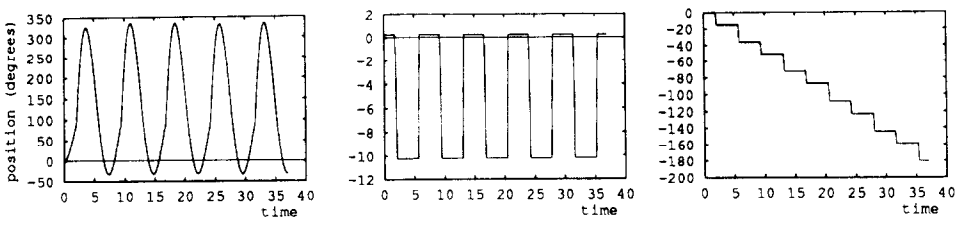


Fig. 2. Motion of escapement mechanism.

The behavioral model generated by my program may be used to numerically simulate a machine's behavior. Fig. 2 shows a plot of a simulation of the behavior of the escapement mechanism using the second dynamic model. The two models, though quite different, both appear to do a good job of predicting the behavior that such a mechanism would actually exhibit. (As a result, simulation output from the first model looks identical to Fig. 2 and is therefore omitted from this article. Both outputs are shown in [15].) My automated modeling algorithms are presented in Section 2.

To the human eye, the data plot in Fig. 2 clearly shows the regularity of the mechanism's behavior, but this regularity is not explicit in the numerical simulation data. Furthermore, though the plot shows a sufficiently long behavior trace to make the behavioral regularity clear, the only reason the simulation ran for that length of time is because of explicit instructions. Also, though Fig. 2 shows the regular behavior of the escapement, even a human would need to do further analysis to decide how long this regular behavior would continue. In Section 3 I present algorithms for continuously processing a stream of simulation data to determine when it has become long enough to show regularities, for characterizing those regularities, and for doing controlled simulated experiments to determine the limits of validity of a hypothesized behavioral regularity.

For the escapement, this process results in a concise description of its long-term behavior which explicitly recognizes the most important qualitative feature of that behavior, its repetitive nature, as well as the most important quantitative feature, the fact that each repetition of a behavior cycle has almost exactly the same duration. Thus my program can start with a description of the shapes of a clock's parts and in effect conclude that the mechanism meets a functional specification for a clock.

Although many mechanical device simulators are commercially available (see references in Section 4), they are incapable of using information about the shape of a machine's parts, so the users of these simulators inherit the following modeling tasks:

- for each pair of parts in permanent contact, completely describe the resulting geometric constraint;
- for each pair of parts which may intermittently come in contact, supply a subroutine which for any positions of the two parts will determine whether or not they are in contact or interpenetrating.

These tasks are both done automatically by my program. Other researchers have addressed aspects of these modeling issues (see Section 4). The problem of extracting a qualitative description of the machine's expected long-term behavior from numerical

simulation output has not been addressed for machines as complex as the escapement described above (see Section 3).

2. Creating behavioral models

My program creates a behavioral model of a machine directly from a representation of the physical structure of the system to be modeled. The representation used[1] is Constructive Solid Geometry (CSG) [24,33], which is widely used by CAD/CAM solid modeling systems in order to represent the shapes of mechanical devices. In the CSG representation, each part in a machine is represented as a closed subset of three-dimensional Euclidean space which is formed by applying the Boolean set operations of union, intersection, and difference to a small set of primitive solids. For example, a square plate with a hole in it might be represented as the difference of block and a cylinder, where the block would be appropriately sized and positioned to represent the plate, and the cylinder would have the correct diameter and position so that the difference operation would create the desired hole in the plate. This geometric representation is supplemented with information about other physical properties such as masses, spring constants, and coefficients of friction. Figs. 12 and 13 in Section 2.3 show the complete input data for the escapement mechanism.

2.1. Kinematics

In order to identify a useful set of state variables, my program must determine the kinematic properties of a mechanism, which requires reasoning about the mechanism's geometry. Kinematic analysis starts with the identification of *kinematic pairs*, pairs of parts which mutually constrain each other's motion. The concept is due to Reuleaux [34], who classified kinematic pairs as either *lower pairs* or *higher pairs*.

When two parts of a mechanism form a kinematic pair, there must be a set of points where the two parts are in contact. If this set of contact points is a two-dimensional surface, then the kinematic pair is classified as a lower pair. Examples of lower pairs include the revolute pair, which limits the relative motion of the elements to rotation about a single axis; the prismatic pair, which allows only relative translation parallel to an axis; and the cylindrical pair, which allows both motions. Only a few lower pairs are geometrically possible. Fig. 3 shows three examples of revolute pairs: the crankshaft/frame pair, the crankshaft/connecting rod pair, and the piston/connecting rod pair. Fig. 4 shows an example of a prismatic pair. The piston/frame pair in Fig. 3 is a cylindrical pair, though in the complete mechanism it behaves as a prismatic pair since it isn't allowed to rotate.

Higher pairs are kinematic pairs in which contact between the elements takes place along lines or points of contact rather than over a full surface. The most common higher pairs are gears.

[1] I use the PADL-2 solid modeling system [20].

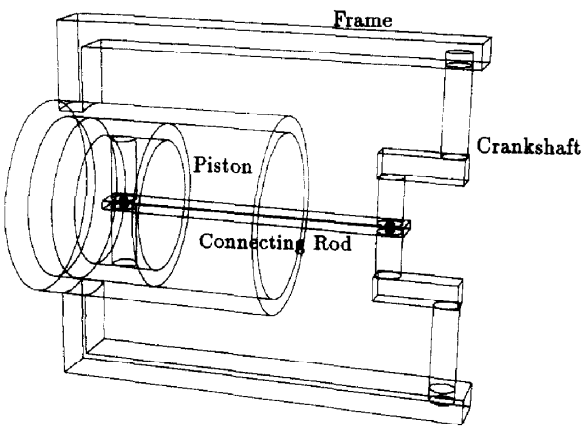


Fig. 3. Crank mechanism.

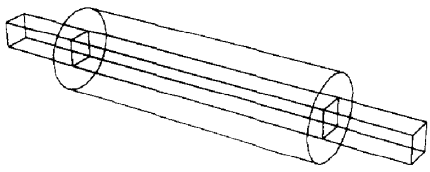


Fig. 4. Prismatic pair.

2.1.1. *Identifying lower pairs*

The central concept used to identify lower pairs is *symmetry*. A three-dimensional shape has symmetry if it can be generated from a two-dimensional shape by a simple operation like translation or rotation. If two parts form a lower pair they must (have subparts which) share a common symmetry: for a revolute pair, rotational symmetry; for a prismatic pair, prismatic (translational) symmetry; for a cylindrical pair, both symmetries. For example, in Fig. 3 the crankshaft forms a revolute pair with the frame, which is only possible if the area of contact between the two parts has rotational symmetry. However, it is important to note that the crankshaft, considered as a whole, does not have rotational symmetry, and neither does the frame. The *subparts* of each which are actually in contact with the other part must have rotational symmetry, though, so the process of detecting lower pairs starts by listing all the symmetries of all the subparts of every moving part of the mechanism.

The Constructive Solid Geometry representation (widely used in CAD/CAM solid modeling systems) makes it easy to compute the list of the symmetries of a part's subparts. CSG primitives are generally regular solids like blocks and cylinders which have clearly defined symmetries. For example, the crankshaft in Fig. 3 is a combination of primitive solids including several cylinders. Every cylinder has rotational symmetry because the cylinder is identical to the volume swept out by rotating a two-dimensional cross-section of the cylinder about the cylinder's axis of symmetry. Every cylinder also

SYMMETRY:
1. symmetry type (rotational or prismatic)
2. the axis of symmetry
3. the list of primitive solids in the part having the symmetry

Fig. 5. The SYMMETRY data structure.

Algorithm 1. *Find all symmetries of all subparts.*
For each *PART*
  For each primitive solid *PRIM* in *PART*
    For each symmetry *SYM* of *PRIM*
      If *SYM* is already on *PART*'s symmetry list
        then add *PRIM* to *SYM*'s list of primitive solids
        else add *SYM* to *PART*'s symmetry list

has prismatic symmetry because all cross-sections perpendicular to its axis of symmetry are identical. Some of the primitive solids forming the crankshaft are blocks, which have no rotational symmetry but have three different prismatic symmetries with respect to three perpendicular axes.

Algorithm 1 generates a list of all the symmetries of all the subparts of every moving part of the mechanism. Each subpart symmetry is represented by an instance of the SYMMETRY data structure (Fig. 5). Algorithm 1 finds five SYMMETRY instances for the crankshaft in Fig. 3—three prismatic, two rotational. The two rotational symmetries have parallel axes but must be considered distinct because rotations about distinct axes produce differing effects even if the axes are parallel. Several primitive solids may participate in the same SYMMETRY instance: for example the rotational symmetry about the axis of the crankshaft/frame kinematic pair applies to two separate cylindrical subparts of the crankshaft. Though the cylinder subparts of the crankshaft have two different rotational symmetries, they all have the same prismatic symmetry since prismatic symmetry does not have a unique axis of symmetry—any line parallel to the axis can also define the symmetry. Of course not all symmetries result in kinematic pairs: for the crankshaft, the two rotational symmetries do but the three prismatic symmetries do not. However, the kinematic pair identification algorithms expect all subpart symmetries as input which can then be filtered to remove irrelevant symmetries.

The symmetry lists are used to find the lower kinematic pairs of the mechanism. The kinematic pairs of the mechanism are represented by a list of KINEMATIC PAIR data structures (Fig. 6). The lower pairs are identified by using Algorithm 2 to find the initial list of kinematic pairs, and then using Algorithm 5 to refine the descriptions of the kinematic pairs on the list.

Algorithm 2 looks for pairs of parts having subparts with a common symmetry, then uses Algorithm 3 to determine whether the subparts form a kinematic pair. The subparts forming a kinematic pair may consist of several primitive solids. For example, in Fig. 3,

KINEMATIC PAIR:

1. type of kinematic pair (revolute, prismatic, cylindrical, or gear, in the current implementation)
2. the two parts forming the kinematic pair
3. the axis of the kinematic pair (for lower pairs)
4. the gear ratio (for gear pairs)
5. a list of pairs of primitive solids making up the kinematic pair
6. a list of CONSTRAINT data structures [15]

Fig. 6. The KINEMATIC PAIR data structure.

Algorithm 2. *Find the initial list of lower pairs.*

For each pair of parts *PART1* and *PART2*
  For each symmetry *SYM* shared by *PART1* and *PART2*
    For each primitive solid *PRIM1* in *PART1* having symmetry *SYM*
      For each primitive solid *PRIM2* in *PART2* having symmetry *SYM*
        Apply Algorithm 3 to *PRIM1* and *PRIM2*.
        If they form a kinematic pair *KP*, then apply Algorithm 4.

Algorithm 3. *Check kinematic pair conditions.*

If *PRIM1* and *PRIM2* satisfy the following conditions:

1. They are the same type of primitive solid
2. One of them is solid and the other is hollow
3. Their dimensions are the same (except possibly along the axis of symmetry)
4. Their intersection is not empty and is also a primitive solid of the same type and dimensions

then they form part of a kinematic pair *KP*

two of the primitive solids in the crankshaft participate in the kinematic pair with the frame. Each primitive solid in one element of the kinematic pair will be paired with a primitive solid of the same type (e.g. block or cylinder) in the other element of the kinematic pair. One of these primitive solids must be solid, and the other must be hollow. A hollow primitive solid is a child of a difference node in a CSG tree. For example, the crankshaft in Fig. 3 fits into two holes in the frame. These holes are hollow primitive solids, formed by subtracting cylinders of the appropriate size and position from the surrounding solid block.

The dimensions of the paired primitive solids must be the same, except along the axis of symmetry. (My algorithms currently neglect the possibility of slippage or play in kinematic pairs.) In the CSG representation, the dimensions of a cylinder are its radius and length, and the dimensions of a block are the lengths of three perpendicular sides. In a revolute pair the solid and hollow cylinders must have the same radius, but

Algorithm 4. *Merge two primitive solids into a kinematic pair.*

```
If KP is not already on the mechanism's list of kinematic pairs
  then add KP to the mechanism's list of kinematic pairs
       If SYM.type == prismatic then KP.type := prismatic
       If SYM.type == rotational then KP.type := revolute
  else add the pair (PRIM1,PRIM2) to KP's list of pairs of primitive solids
       If     KP.type == prismatic AND SYM.type == rotational
         OR KP.type == revolute  AND SYM.type == prismatic
         then KP.type := cylindrical
```

their lengths may differ—for example, in the crank mechanism the solid cylinders of the crankshaft extend far beyond the length of the hollow cylinders they fit into. In a prismatic pair, the hollow and solid primitives must have identical cross-sections, but again they may differ in the dimension along the axis of symmetry, as is the case for the prismatic pair in Fig. 4. Finally, the two primitive solids must have a non-empty intersection. A hole in one place will not form a kinematic pair with a solid that is somewhere else on the same axis. The intersection may however be smaller than either of the two primitive solids: the solid element may extend out of the hollow, as in the crankshaft/frame pair, and/or be inside a much larger hollow element, as is the case for the piston/frame pair.

Since the Constructive Solid Geometry representation is based on Boolean set operations, there is an easy way to define the shapes of two parts in order that they will fit together to form a lower pair: simply describe the shape of the solid element of the pair and then use a single difference operation to create the hollow element by subtracting the solid element from some other (larger) solid body. Algorithm 3 is particularly suitable for identifying lower pairs whose elements are defined in this way. However, in some cases the elements of a kinematic pair may be specified in other ways that are less convenient for the user but which yield exactly the same shapes, since Constructive Solid Geometry representations for solids are not unique [33]. For example, a hole with a rectangular cross-section may be created either as a difference of two blocks or as a union of four blocks. A limitation of Algorithm 3 is the underlying assumption that the Boolean set operation used to form the hollow element of a kinematic pair is a difference operation, so Algorithm 3 could not currently identify a kinematic pair whose hollow element was formed as the union of four blocks.

If two primitive solids satisfy the conditions in Algorithm 3, then Algorithm 4 is used to instantiate a new kinematic pair or merge new data into a previously instantiated pair. For example, a cylindrical pair has both prismatic and rotational symmetry. Depending on which symmetry is processed first, the pair will initially be instantiated as either prismatic or revolute, and then when the other symmetry is processed the type of kinematic pair will be set to cylindrical.

Often the initial list of kinematic pairs will include degrees of freedom which are actually unavailable to the mechanism being analyzed. Algorithm 5 is a heuristic algorithm which filters the degrees of freedom associated with each kinematic pair, attempting to

Algorithm 5. *Refine the list of kinematic pairs.*
For each kinematic pair *KP*
If *KP*.type == cylindrical and its translational motion is blocked in both directions
then *KP*.type := revolute
If *KP*.type == cylindrical and one of its primitive solids is a block
then *KP*.type := prismatic
If *KP*.type == cylindrical and it has two cylindrical subparts which are not coaxial
then *KP*.type := prismatic

identify unavailable degrees of freedom which it can then remove from the list. Any kinematic pair with a cylindrical subpart will have both rotational and prismatic symmetry and therefore will initially be labeled a cylindrical pair, but often a further limitation of the degrees of freedom is possible. If translational motion is blocked because both ends of the solid element of the pair are in contact with solid parts of the other element, then the translational degree of freedom can be removed, and the kinematic pair can be declared a revolute pair. If translation is possible in at least one direction, though, even for a short distance, the translational degree of freedom is kept, since my algorithms currently neglect the possibility of slippage or play in kinematic pairs. If rotational motion is blocked because the total cross-section of the pair doesn't have rotational symmetry, the kinematic pair can be declared a prismatic pair. Every additional degree of freedom eliminated in kinematic analysis will allow simulation of the machine's dynamics to run more efficiently, since fewer state variables will be needed.

Algorithm 5 only removes one of a cylindrical pair's two degrees of freedom if it is able to determine that that degree of freedom will never be available to the kinematic pair. Therefore a degree of freedom that is actually available to a kinematic pair will never mistakenly be removed, but, on the other hand, there may be cases when a degree of freedom will not be removed by Algorithm 5 even though it legitimately could be. The reason for this cautious strategy is that missing degrees of freedom may result in completely wrong behavioral simulations, but extra degrees of freedom will still give correct, though less efficient, simulations. The final behavioral model of a machine incorporates the geometric model, and in the behavioral model any contact between solid objects will generate an appropriate force. Thus if, for example, a revolute pair were classified as cylindrical, the fact that its translational motion was blocked would still be implicitly represented in the behavioral model, and in a behavioral simulation the kinematic pair would in fact behave as a revolute pair because any translational motion would be resisted by an appropriate force. On the other hand, an unwarranted reduction of degrees of freedom may result in incorrect simulated behavior because the behavioral model will have missing state variables, and forces which would in reality influence the missing state variables will have no effect in this model.

2.1.2. Identifying higher pairs

At present, the only higher pairs my kinematic analyzer knows about are gear pairs. However, my model generator still works on mechanisms including other higher pairs:

Algorithm 6. *Find gear pairs.*
For each pair of parts *PART1* and *PART2*
  For each primitive solid *PRIM1* in *PART1* which is a "gear"
    For each primitive solid *PRIM2* in *PART2* which is a "gear"
      If *PRIM1* intersects *PRIM2*
        then instantiate a new "gear" kinematic pair

since their behavior arises from the physics of bodies in contact, the dynamics algorithms in Section 2.2.1 and Appendix A allow that behavior to be modeled and simulated. Fig. 14 shows an example of such a higher pair.

Modeling the physical structure of a gear poses a difficulty for the CSG representation described in Section 2.1, since gear teeth typically have a profile curve such as a cycloid or an involute which cannot be represented with the primitives available in most CSG implementations. My program expects a gear to be represented by its motion envelope, which is a cylinder, labeled with the property "gear". This convention is just for shape representation and does not imply that the object modeled actually functions as a gear. If the motion envelopes of two such "gears" overlap in space, then they are considered to be meshed.[2] My program does not presently model gear tooth geometry, so an arbitrarily small overlap is considered sufficient for meshing, and the number of teeth is considered to be directly proportional to gear diameter, so the kinematic analyzer computes the gear ratio to be the ratio of the diameters of the gears. A simple extension that would be a compromise between the present scheme and a full representation of tooth geometry would be to annotate each gear with the number of teeth and the tooth depth. Gear pairs are represented by the KINEMATIC PAIR data structure (Fig. 6) and identified by Algorithm 6.

2.1.3. Partitioning

The goal of kinematic analysis is to choose a useful set of state variables. It is desirable that the set of state variables be as small as possible since simulations will run more efficiently with fewer state variables. State variables are chosen by partitioning the kinematic pairs into kinematic subsystems, sets of parts which are kinematically constrained so that the entire set has only a single degree of freedom. For example, the positions and velocities of every gear in a gear train can be computed by knowing the position and velocity of any one of the gears, so it is preferable to have only one degree of freedom associated with the entire gear train.

Each kinematic subsystem has two state variables associated with it: a position variable and a velocity variable. My current partitioning algorithm is fairly limited, and requires each moving part to be part of a fixed-axis kinematic pair. Therefore the kinematic pairs that bind kinematic subsystems together must be higher pairs. In this section I will discuss partitioning for kinematic subsystems bound together by gears. For example,

[2] This is the only case of input to the kinematic analyzer in which the CSG representations of solid bodies are allowed to overlap in space.

KINEMATIC SUBSYSTEM:

1. the fixed-axis KINEMATIC PAIR
2. the motion envelope of the moving part of the KINEMATIC PAIR
3. a list of INTERACTOR data structures (Fig. 9)
4. the inertia of the moving part
5. the linear damping coefficient of the moving part
6. the spring description
7. the children: a list of KINEMATIC SUBSYSTEM data structures
8. the parent: a KINEMATIC SUBSYSTEM data structure
9. the type of relationship with the parent
10. the relationship data

Fig. 7. The KINEMATIC SUBSYSTEM data structure.

⟨Escape Wheel⟩ ⟶ ⟨Lever⟩ ⟶ ⟨Frame⟩ ⟨Balance⟩

a) Only balance moving (e.g. Fig. 1(a) and Fig. 1(d))

⟨Balance⟩ ⟶ ⟨Lever⟩ ⟶ ⟨Escape Wheel⟩

b) All three parts moving together (e.g. Fig. 1(b) and Fig. 1(c))

[Note: these are for the *second* behavioral model, Appendix A]

Fig. 8. Kinematic subsystem trees for escapement.

in the chiming clock shown in Fig. 26 (Section 3.2.2) the escape wheel and the large gear form a single kinematic subsystem. In Appendix A I will introduce several transient higher pairs that form temporary kinematic subsystems so that, for example, in Fig. 1(c) the three moving parts are considered to be temporarily joined by two transient higher pairs into a single kinematic subsystem which has a total of one degree of freedom.

A kinematic subsystem is a set of elementary kinematic subsystems, kinematic subsystems with only one moving part, each represented by a KINEMATIC SUBSYSTEM data structure (Fig. 7). This data structure includes data from the model of physical structure (see Section 2.1): the inertia and damping of the moving part and a description of the spring driving it (if there is one). The data structure also includes pointers to the node's parent and children, and these pointers organize the kinematic subsystem as a tree with a particular elementary kinematic subsystem as its root, though the tree may be reorganized at any time using Algorithm 21 (Section A.2.1) to make a different elementary kinematic subsystem be the root. Fig. 8 shows kinematic subsystem trees for escapement when the moving parts are linked by the transient higher pairs of Appendix A. For example, Fig. 8(a) indicates that the position of the escape wheel is determined by that of the lever, and the position of the lever is determined by that of the frame. Since the frame never moves, neither do the lever or escape wheel. The balance, however, is in a separate kinematic subsystem and can move. In Fig. 8(b), all three

Algorithm 7. *Form permanent kinematic subsystems.*

1. For each kinematic pair *KP*
   If one element of the pair is the frame
   then instantiate a new KINEMATIC SUBSYSTEM data structure *KS*
   Store the value *KP* in the KINEMATIC PAIR field of *KS*
   Fill in physical structure data from the input model
   Set all other fields to NULL
2. For each gear pair *GP*
   If both of the associated kinematic subsystems have parents
   then make one the root of its tree by reversing links
   Make one of the associated kinematic subsystems which is a root the child of the other kinematic subsystem
   Set the relationship type to GEAR
   Set the relationship data to the gear ratio

INTERACTOR:

1. the two KINEMATIC SUBSYSTEMS that interact
2. the two sets of interacting primitive solids
3. the ACTIVE flag (see Appendix A)

Fig. 9. The INTERACTOR data structure.

parts can move, and the position of the escape wheel determines that of the other two. These trees are shown only for the second behavioral model (Appendix A) because for the first model, the three escapement subsystems always remain separate and never join to form a tree.

The position and velocity state variables for a kinematic subsystem are the position and velocity of the elementary kinematic subsystem which is the root of the tree. The states of the other elementary kinematic subsystems may be computed by descending the tree, computing the state of each node from the state of its parent. Algorithm 7 generates all elementary kinematic subsystems, and forms permanent links between each pair of elementary kinematic subsystems whose moving parts are gears which are meshed with each other. In Appendix A I will describe dynamic partitioning algorithms which form additional links when the moving parts of different kinematic subsystems temporarily form certain other higher pairs.

2.1.4. Interaction analysis

Interaction analysis determines which of the kinematic subsystems might potentially come in contact during the motion of the mechanism. Potential interactions are represented by the INTERACTOR data structure (Fig. 9). This preprocessing results in an optimization of the model, but it isn't an essential part of the model-generation process. Without preprocessing, any pair of parts would have to be treated as potentially interacting, which would slow down simulations using the model. My interaction analysis

works by checking for intersecting motion envelopes; the algorithms are described in [15] and are currently limited to fixed-axis mechanisms.

2.2. Dynamics

After finding a useful set of state variables, my program must complete the behavioral model of a mechanism by determining how the state variables change with time, which is described using differential equations

$$\frac{ds_i}{dt} = g_i(s, t) \tag{1}$$

where the s_i are state variables. The functions g_i are part of the behavioral model of the mechanism that my program creates. The behavior of a mechanism depends on the forces acting between the parts of the mechanism. By making different sorts of simplifying assumptions, my program can model these forces in two different ways: a local model in which the force between two parts depends only on the state of those two parts, and a global model in which the forces between two parts are determined by simultaneous equations involving the whole mechanism. Section 2.4 compares the two models.

2.2.1. Local model for contact forces

Consider a mechanism consisting of n elementary kinematic subsystems, kinematic subsystems with only one moving part. Let x_i be the position variable associated with the ith elementary subsystem, and let v_i be the time derivative of x_i. The state of the machine at any time is specified completely by the values of the $2n$ state variables x_i and v_i, and the way the state changes is described by $2n$ differential equations, each an instance of Eq. (1). For mechanical devices, these differential equations are typically nonlinear and in fact may not be expressible in closed form, so in general they cannot be solved explicitly. To solve these equations numerically requires algorithms to compute the time derivatives of the state variables. For the x_i this computation is trivial since

$$\dot{x}_i = \frac{dx_i}{dt} = v_i. \tag{2}$$

For the v_i

$$\dot{v}_i = \frac{dv_i}{dt} = \frac{f_i}{I_i}. \tag{3}$$

In this equation I_i is the *inertia* of the moving part. If the moving part forms a prismatic pair with the frame, then the inertia is just its mass, and if it forms a revolute pair with the frame then the inertia is the moment of inertia of the moving part about the axis of the revolute pair. The inertia is available from the KINEMATIC SUBSYSTEM data structure (Fig. 7). For a prismatic pair, f_i, the *force* on the moving part, is just the net force in the traditional sense, while for a revolute pair it is the net torque.

In my current implementation contributions to the total force on a subsystem come from three sources: springs, friction, and contact forces. Springs are attached between

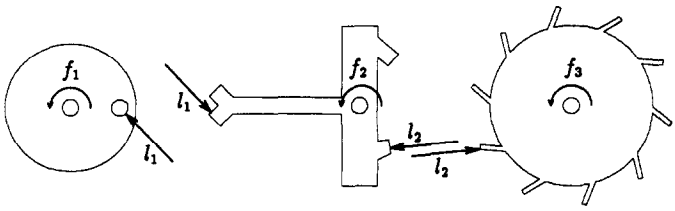


Fig. 10. Free-body diagrams of escapement moving parts.

the two elements of an elementary kinematic subsystem, and the spring force is a linear function of the position state variable. For example, the escapement mechanism in Fig. 1 involves two springs: the mainspring which gives the energy to run the clock, and the hairspring which makes the rotation of the balance oscillate at a regular rate. In my current implementation springs are represented by annotating the input to the model generator with information about springs constants; the shapes of the springs are not explicitly represented. Friction occurs between the two elements of an elementary kinematic subsystem, and the frictional force is a linear function of the velocity state variable (i.e. viscous damping, not Coulomb friction).

The model for contact forces which I discuss in the present section explicitly considers the small elastic and plastic distortions of the parts near the point of contact. I use a model which greatly simplifies the physics involved but is still quite useful. Bodies are modeled as being rigid, but their volumes are allowed to overlap in space, and this overlap gives rise to a force. If o is the depth of overlap, the magnitude l of the force is defined by

$$l = \begin{cases} Eo + D\dot{o}Mo, & \text{if } o \geqslant 0 \text{ and } E + DM\dot{o} \geqslant 0, \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

The contact force is basically being modeled as a linear spring with linear damping, where E is the spring constant and D is the damping coefficient. The additional factor Mo in the second term on the right is there to make the force function continuous. (Typically $\dot{o}$ has significant magnitude even when $o = 0$, so without Mo, the damping force would jump discontinuously from zero to a significant finite value at the first instant of contact.) The direction of the force is taken to be that of the surface normal at the point of contact. Simulations of mechanism behavior appear fairly insensitive to changes in the values of the constants in Eq. (4), and the values

$$E = 100000, \qquad D = 10000, \qquad M = 100,$$

are used for all examples in this article.

The total force f_i on subsystem i is

$$f_i = -k_i x_i - h_i v_i - \sum_{k \in \text{contacts}} \alpha_{ik} l_k, \tag{5}$$

where k_i is the spring constant of the spring, if any, attached to subsystem i, h_i is the coefficient of linear friction of subsystem i, l_k is the contact force described above

Algorithm 8. *Compute time derivatives of state variables.*

For each elementary kinematic subsystem i
  Set $\dot{x}_i = v_i$ and $\dot{v}_i = (-k_i x_i - h_i v_i)/I_i$
For each INTERACTOR data structure (Fig. 9)
  For each pair of potentially interacting primitive solids
    If Algorithm 9 determines that their volumes overlap in space
      Let o be the depth of overlap
        $\boldsymbol{n}$ be the normal vector at the point of contact
      For each of the two interacting elementary kinematic subsystems i
        Let $\hat{\boldsymbol{a}}_i$ be a unit vector along the axis of the kinematic pair
        If subsystem i is a prismatic pair
          then set $\alpha_i = \boldsymbol{n} \cdot \hat{\boldsymbol{a}}_i$
        Else
            Let $\boldsymbol{r}_i$ be the radius vector from the axis to the point of contact
            Set $\alpha_i = \boldsymbol{r}_i \times \boldsymbol{n} \cdot \hat{\boldsymbol{a}}_i$
        If $\boldsymbol{n}$ points away from subsystem i, then negate α_i
      Let $\dot{o} = -(\alpha_1 v_1 + \alpha_2 v_2)$
        $l = Eo + D\dot{o}Mo$
      For each of the two interacting elementary kinematic subsystems i
        Subtract $\alpha_i l/I_i$ from $\dot{v}_i$

between the two parts in contact at contact k, and α_{ik} is the geometric force multiplier taking the contact force l_k into a force on subsystem i. (α_{ik} is zero if subsystem i is not involved in contact k.)

Fig. 10 shows free-body diagrams of the three moving parts of the escapement mechanism, for the state shown in Fig. 1(c) in which the escape wheel is pushing the lever which in turn is pushing the balance. A contact force contributes to $\dot{v}_i$ for two different subsystems. l will be the same for both subsystems since the force acts equally though in opposite directions on each of them, but the factors α will be different. For the escapement state of Fig. 10, the net torque f_1 on the balance is positive, giving the balance a counterclockwise acceleration, while the net torques f_2 and f_3 on the lever and escape wheel are negative, so they both experience clockwise acceleration.

To turn these equations into a useful behavioral model of a mechanism requires algorithms for determining which parts are in contact and for computing the depth of overlap o, the rate of change of overlap $\dot{o}$, and the geometric force multiplier α.

Algorithm 8 computes the time derivatives of the state variables using the equations above.[3] Following Eq. (2), $\dot{x}_i$ is set to be v_i for all elementary kinematic subsystems i.

[3] Though Algorithm 8 appears complicated, it is actually far simpler than a proper model using true elasticity, etc. The equations of elasticity are partial differential equations, and the automating the discretization (i.e. grid or mesh generation) needed to for their computational solution would be quite a challenging task for complex shapes like the escapement mechanism in Fig. 1. While a deeper model might be useful for tasks like predicting part failures by estimating metal fatigue, the added complexity seems hard to justify simply for predicting behavior under normal operating conditions.

Using Eq. (3) and part of Eq. (5), each $\dot{v}_i$ is set to $(-k_i x_i - h_i v_i)/I_i$ to account for the spring and frictional forces. Then Algorithm 9 is used to find parts in contact, Eq. (4) is used to compute the resulting contact force, and this is used in Eq. (5) to find the net $\dot{v}_i$. In the case of a prismatic pair, the geometric force multiplier α will be simply $\boldsymbol{n} \cdot \hat{\boldsymbol{a}}$; it will select the component of the force along the axis of translation. For a revolute pair, α will be $\boldsymbol{r} \times \boldsymbol{n} \cdot \hat{\boldsymbol{a}}$ since the torque due to a force $\boldsymbol{f}$ is $\boldsymbol{r} \times \boldsymbol{f}$ and in this case the force is $l\boldsymbol{n}$ where l is the magnitude of the contact force computed from Eq. (4) and $\boldsymbol{n}$, the surface normal, is the direction of the contact force acting on the body touching the surface. The direction of the force on the other body is $-\boldsymbol{n}$.

The rate of change of overlap $\dot{o}$, which appears in Eq. (4), is the sum of the motions of the two parts at the point of contact, and these motions are computed from the velocity state variables using the same geometric factor α. The fact that both force and velocity transformation use the same geometric factor is basically the principle of the lever. A force at one end of a lever appears at the other end multiplied by a certain factor, and the velocity appears divided by exactly the same factor. This equivalence can be proven more formally by permuting the scalar triple product $\boldsymbol{r} \times \boldsymbol{n} \cdot \hat{\boldsymbol{a}}$ to get $\hat{\boldsymbol{a}} \times \boldsymbol{r} \cdot \boldsymbol{n}$ which will take the angular velocity vector $\hat{\boldsymbol{a}} v_i$ into the component of linear velocity in the surface normal direction.

To compute the net force on each part the time derivative routine must determine which primitive solids have overlapping volumes. For each overlap detected, both the depth of overlap and the surface normal at the point of contact must be computed. Algorithm 8 applies Algorithm 9 to each pair of potentially interacting primitive solids to compute this information. In some sense Algorithm 9 is actually unnecessary: the PADL-2 solid modeler mentioned in Section 2 includes much more general algorithms for computing solid-body intersections. In my experiments I have found, however, that these general algorithms runs several orders of magnitude slower than Algorithm 9 and would make impossible the real-time simulations achieved by my current implementation (see Fig. 22).

Algorithm 9 is currently limited to "$2\frac{1}{2}$-dimensional" overlapping parts: it works by analyzing two-dimensional projections of three-dimensional situations. Therefore primitive solids can't have arbitrary orientations; they must have prismatic symmetry with respect to the z-axis. Overlap computations are currently limited to interactions between blocks and other blocks or cylinders. All these limitations could be removed, of course, simply by using the slow but general intersection capabilities of PADL-2.

For the model of contact forces used in this section to be plausible, parts may only overlap slightly. Thus the algorithms in this section compute depths of overlap based on the assumption that the overlap will be slight. However, if the simulator is adjusting its step size to try to find an optimum, it may start by trying incorrect sizes that lead it to project physically implausible states like those shown in Fig. 11(c) and Fig. 11(e). To allow the simulator to make proper adjustments, the depth values returned in these cases must be a reasonable extrapolation of the results in the physically plausible range.

In the case of two potentially overlapping blocks, Algorithm 10 looks for a face of one of the blocks which separates one corner of the other block from its other three corners, with the one corner being inside the first block and the other three corners

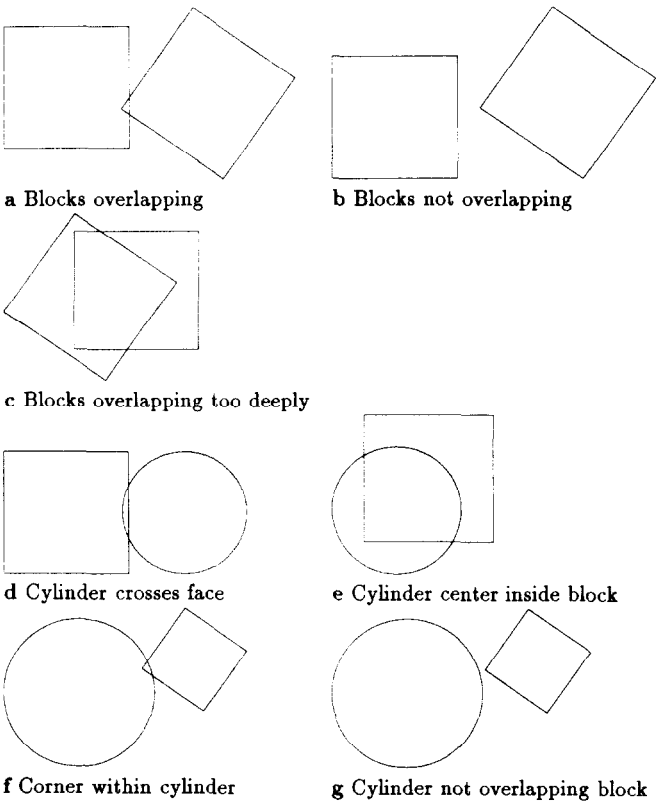


Fig. 11. Cases the overlap detector must deal with.

Algorithm 9. *Check overlap of two primitive solids.*

1. If the projections of the solids on the z-axis do not intersect, then exit
2. Project the solids on the x–y plane and apply Algorithm 10 or Algorithm 11
3. Return the depth of overlap, the point of overlap, and the local surface normal

being outside. See Fig. 11(a). Alternatively, if one of the blocks has a face which is in a plane such that one block is entirely on one side of the plane and the other block is entirely on the other side, then the blocks do not overlap. See Fig. 11(b). Otherwise the blocks do overlap, but more deeply than physically plausible, so the depth must be extrapolated. See Fig. 11(c).

Algorithm 11 deals with the case of a cylinder which potentially overlaps a block. A cylinder may cross a face of a block, as in Fig. 11(d), or contain a corner of a block, as in Fig. 11(f). If the block contains the center of the cylinder, the depth must be extrapolated from the physically plausible range. See Fig. 11(e). Otherwise the solids do not overlap. See Fig. 11(g).

Algorithm 10. *Check overlap of two blocks.*

1. Check each of the four corners of each of the two blocks to see if it is the only corner within the other block, as shown in Fig. 11(a)
2. If a corner is within the other block, return the corner as the point of contact, the distance from the nearest face as the depth of overlap, and the perpendicular out of the face as the local surface normal
3. If one of the four faces of either block separates the two blocks, as shown in Fig. 11(b), then exit
4. The blocks overlap deeply as in Fig. 11(c). Choose the point midway between the blocks' centers as the point of contact, the overlap between the blocks' containing spheres as the depth of overlap, and let the "normal" be along the line between the blocks' centers

Algorithm 11. *Check overlap of a block and a cylinder.*

1. If the cylinder crosses a face of the block as shown in Fig. 11(d), return the most deeply embedded point of the cylinder as the point of contact, the distance from the nearest face as the depth of overlap, and the perpendicular out of the face as the local surface normal
2. If the block contains the center of the cylinder as in Fig. 11(e), choose the point midway between the solids' centers as the point of contact, the overlap between the cylinder and the block's containing sphere as the depth of overlap, and let the "normal" be along the line between the solids' centers
3. Check each of the four corners of the block to see if it is within the cylinder, as shown in Fig. 11(f)
4. If a corner is within the cylinder, return the deepest corner as the point of contact, the distance to the surface as the depth of overlap, and the normal to the cylinder's surface
5. The solids don't overlap (Fig. 11(g)). Exit

2.2.2. *Global model for contact forces*

My program can also create an alternative model for contact forces that represents machines as collections of perfectly rigid bodies and which treats interactions between different kinematic subsystems by forming temporary kinematic pairs that join the various interacting subsystems together into a single subsystem. When new temporary kinematic pairs are formed, the behavioral model dynamically reduces the number of state variables, because the state of each kinematic subsystem is determined by just two state variables—a position and a velocity—no matter how many parts are in the subsystem. When parts separate, the number of state variables increases again. In this model, the forces shown in the free-body diagrams in Fig. 10 are still acting, but they never need to be explicitly considered in predicting the motion of the mechanism: their effects are all captured by the kinematic constraints which reduce the number of state variables. Though the forces do not need to be computed to predict a machine's behav-

ior, if desired they may be determined using simultaneous equations involving the whole mechanism, which is why I call this a global model for contact forces. The details of this model are given in Appendix A.

2.3. Simulation

A computational simulation of a machine's behavior may be generated by numerically solving the differential equations of its behavioral model for a specific set of initial conditions. The differential equations in my local model for contact forces are "stiff", which means that standard algorithms for numerically solving differential equations will be unstable unless very small time steps are taken [12]. Thus for simulations based on that model I use a special algorithm for stiff differential equations, implemented in the publicly-available subroutine GEAR [23]. My global model for contact forces does not have stiff equations, so for simulations based on that model I use a more standard algorithm, the Runge–Kutta–Fehlberg method, implemented in the publicly-available subroutine RKF45 [38]. Both GEAR and RKF45 are available via electronic mail from `netlib@ornl.gov`.

For both models the numerical solvers must be carefully controlled to ensure that they never take time steps large enough to allow two moving parts of a mechanism to pass through each other. For example, consider the escapement mechanism as shown in Fig. 1(b), where the balance collides with the lever. Before the collision, the motion of the balance is very smooth, and the simulator can predict its position very accurately even if it takes quite large time steps. However, if a simulator takes too large a time step it may go directly from a state in which the balance is on one side of the lever to a state in which the balance is on the other side of the lever, so the simulation will fail to predict the collision of the balance with the lever.

Figs. 12 and 13 show the complete input data given to my program for the escapement mechanism in Fig. 1. Fig. 12 shows the geometric description of the mechanism which is parsed by the PADL-2 solid modeling program [20]. Fig. 13 shows the supplementary input data indicating that the moments of inertia of the lever, balance, and escape wheel are 10, 20, and 200, respectively, that their coefficients of friction with the frame are 5, 6, and 20, and that a spring with spring constant 15 and relaxed position at 90 degrees is attached between the balance and the frame and a spring with spring constant 20 and relaxed position at −10000 degrees is attached between the escape wheel and the frame.

Initial conditions must be specified to give a complete initial value problem for the numerical methods to solve. For the following examples the initial conditions are that at time = 0 the parts of the mechanism are not moving (all velocities = 0), and are still in the original positions specified in the input model. Fig. 2 in the introduction shows the behavior of the escapement mechanism (Fig. 1) when simulated with the global model for contact forces described in Appendix A. A simulation using the local model yields identical behavior, though it runs more slowly. (The global model is fast enough to permit real-time simulation on a workstation.) Fig. 22 at the end of this section gives timing data for all the examples.

Fig. 14 shows a mechanism which converts rotary motion into reciprocating motion. It consists of a wheel and an arm which both form revolute pairs with the frame. A

```
generic esc(esc)
tooth = blo(x=0.3,y=5.4) at (movx=-3.2)
gap = 36
ewheel = ((cyl(d=10,h=1) un tooth un\
 (tooth at (degz=gap)) un\
 (tooth at (degz=2*gap)) un\
 (tooth at (degz=3*gap)) un\
 (tooth at (degz=4*gap)) un\
 (tooth at (degz=5*gap)) un\
 (tooth at (degz=6*gap)) un\
 (tooth at (degz=7*gap)) un\
 (tooth at (degz=8*gap)) un\
 (tooth at (degz=360-gap))) dif cyl(d=1)) at (movx=0)
lever = (((((blo(x=10,y=2.2) at (movx=-5,movy=6.4)) un\
 (blo(x=0.9,y=2) at (degz=7,movx=-3,movy=5.6)) un\
 (blo(x=0.9,y=2) at (degz=25,movx=-3,movy=5.6)) un\
 (blo(x=1,y=2) at (movy=0.1,degz=-40,movx=2.4,movy=5.7)) un\
 (blo(x=1,y=8) at (movx=-.5,movy=8)) un\
 (blo(x=1,y=1.8) at (movy=-1,degz=-45,movy=16.2)) un\
 (blo(x=1,y=1.8) at (movy=-1,movx=-1,degz=45,movy=16.2))) dif\
 (cyl(d=1) at (movy=7.5))) at (movy=-7.5)) at (degz=5)) at\
  (movy=7.5)
bwheel = (((cyl(d=8,h=1) at (movy=20,movz=-1)) un\
 (cyl(d=1,h=2) at (movx=-3,movy=20,movz=-1))) dif (cyl(d=1) at\
  (movy=20,movz=-1))) at (movx=0)
frame = ((((blo(x=13,y=31) at (movx=-6.5,movy=-6.5,movz=-2)) un\
 (cyl(h=4) at (movz=-2)) un (cyl(h=4) at (movy=7.5,movz=-2)) un\
 (cyl(h=3) at (movy=20,movz=-2))) dif\
 (ewheel un lever un bwheel)) un\
 (cyl(d=1,h=3) at (movx=-1.6,movy=14,movz=-2)) un\
 (cyl(d=1,h=3) at (movx=1.6,movy=14,movz=-2))) at (movx=0)
attr frame : frame = 1
esc = ewheel asb lever asb bwheel asb frame
```

Fig. 12. Geometric input data for escapement mechanism.

```
< LEVER : FRAME > 10 5
< BWHEEL : FRAME > 20 6 SPRING 90 15
< EWHEEL : FRAME > 200 20 SPRING -10000 20
```

Fig. 13. Supplementary input data for escapement mechanism.

small cylinder which protrudes from the wheel fits in a grove in the arm. As the wheel rotates, the arm is forced to oscillate back and forth. Since my kinematic analyzer only recognizes very common higher kinematic pairs, it does not recognize the pair formed by the wheel and the arm. Nevertheless, my behavioral simulator is able to simulate the

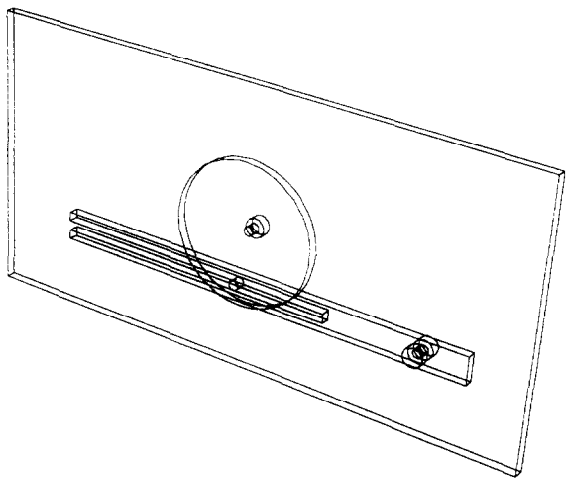


Fig. 14. Mechanism to convert rotary motion into reciprocating (rocking) motion. (Note: spring attached to wheel is present in model but not shown in figure.)

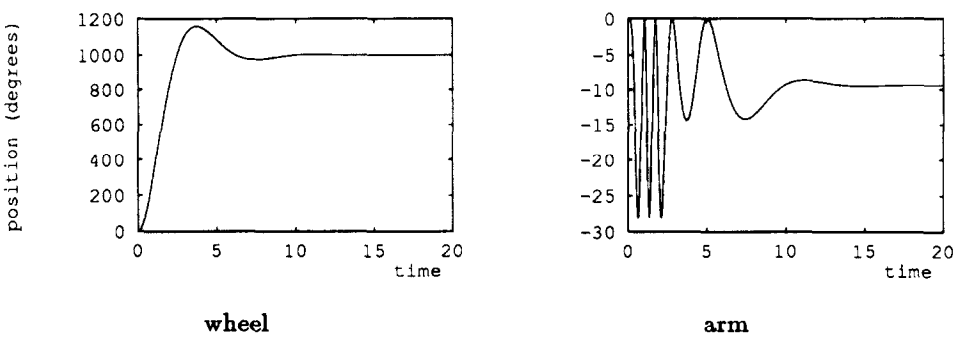


Fig. 15. Motion of the conversion mechanism.

behavior of this mechanism, because the continuous contact between the two elements of the higher pair can be modeled using the same methods that work for intermittent contacts between parts in a mechanism like the escapement. Fig. 15 shows a plot of the results of a simulation of this mechanism, where the wheel is driven by a spring and the mechanism starts in a state with all velocities zero and the parts in the position shown in Fig. 14. (Note: my input representation (Section 2) allows the specification of coefficients of friction, and for all the examples I have specified nonzero friction between the moving parts and the frame.)

Fig. 16 shows a "double escapement" mechanism which is not particularly useful but provides an interesting test for the simulator. Fig. 17 shows a plot of simulation results for the double escapement. It turns out that the double escapement doesn't jam or behave in any unusual way; the two balances and lever move synchronously, and the mechanism exhibits the same regular behavior as the standard escapement.

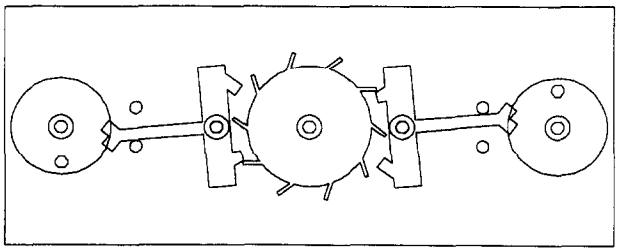


Fig. 16. "Double escapement". (Note: hairsprings attached to balances and mainspring attached to escape wheel are present in model but not shown in figures.)

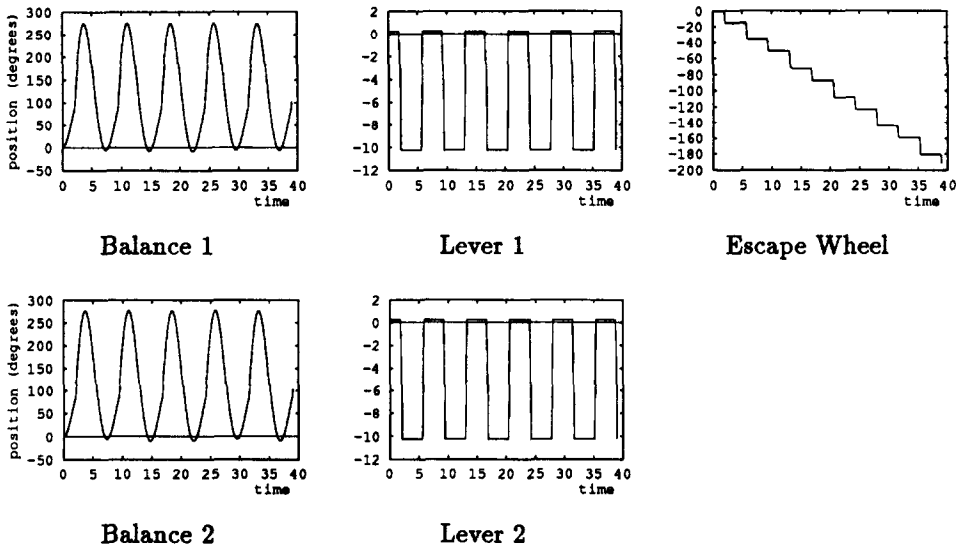


Fig. 17. Motion of the double escapement mechanism.

Fig. 18 shows a simple ratchet mechanism. When a counterclockwise force is applied to the wheel, its motion is blocked by the pawl, as shown in Fig. 19, but when a clockwise force is applied to the wheel, it turns freely. Fig. 20 shows a plot of the results of a simulation of this mechanism with a clockwise force on the wheel. Fig. 21 shows a plot of the results of a simulation of this mechanism with a counterclockwise force on the wheel. The configurations of the mechanisms at the end of the two simulations are shown in Fig. 19.

2.4. *Comparing the models*

The global model for contact forces described in Appendix A has also been implemented and run on most of these examples. The capability to experiment with two disparate models is a useful tool for investigating how choices of approximations and

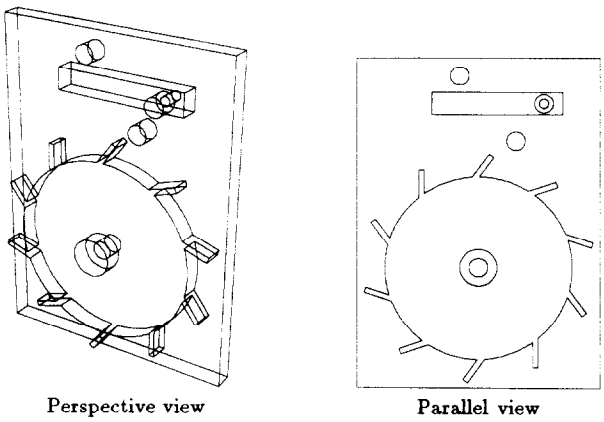


Fig. 18. Ratchet mechanism. (Note: springs attached to wheel and pawl are present in model but not shown in figure.)

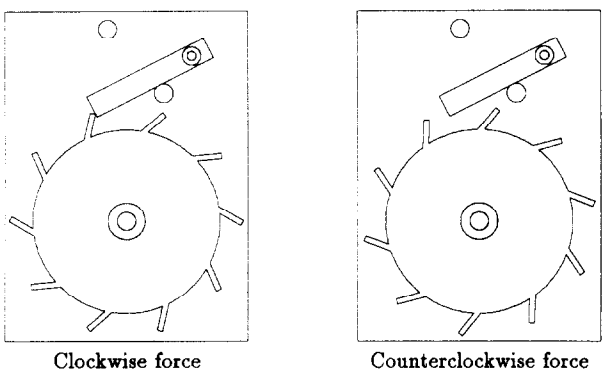


Fig. 19. Final state of ratchet mechanism.

simplifying assumptions influence the forms of behavioral models and the sorts of behavior the models predict. Both of the models appear to give quite realistic behavior predictions for clockwork mechanisms, so simulations using either model are very similar. Fig. 22 shows some experimental data. Note that the global model, at least with this implementation, runs more quickly because it both requires fewer time steps and takes less time to execute each step. This difference is plausible because in the global model, when parts come in contact so that one part pushes another, the behavioral model dynamically reduces the number of state variables, which speeds the solution of the differential equations. Also, the solution of stiff differential equations like those in the local model tends to take somewhat longer even when using a solver tailored for stiff problems.

For an application like routine design of some type of mechanical device, the global model seems superior to the local model for the task of evaluating performance of

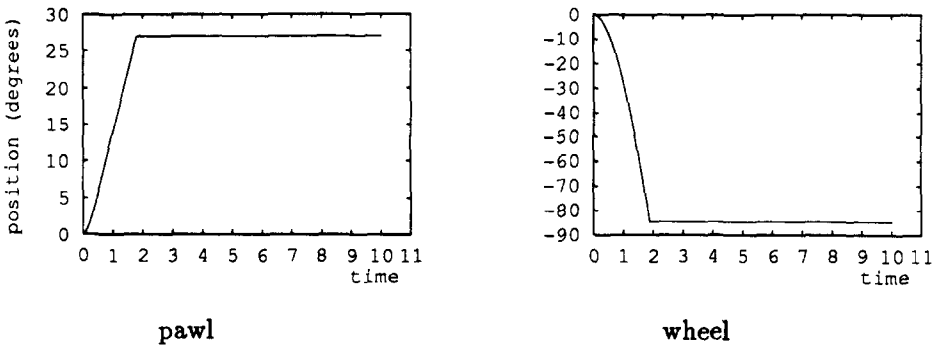


Fig. 20. Motion of the ratchet mechanism (clockwise force).

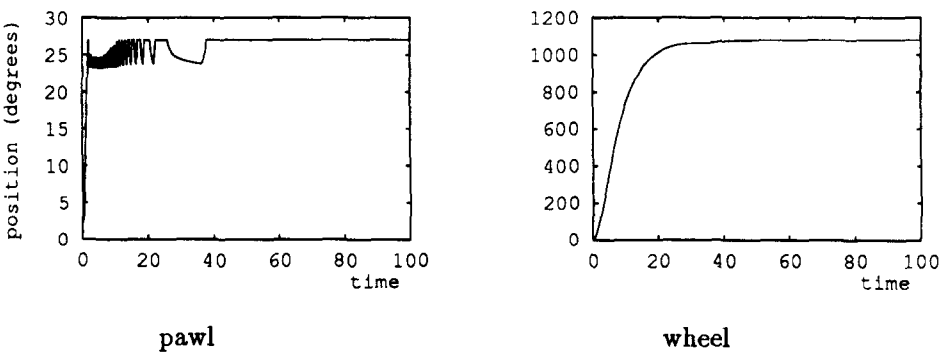


Fig. 21. Motion of the ratchet mechanism as a function of time (counterclockwise force).

	Execution time (seconds)	Steps	Time/step (ms)	Simulated time
Local model				
Escapement	49.2	3942	12.5	37
Rotary to Reciprocation	2.5	2119	1.2	20
Ratchet (cw)	0.82	594	1.4	10
Ratchet (ccw)	16.7	7928	2.1	100
Double Escapement	181	5073	36	39
Global model				
Escapement	1.3	836	1.6	37
Rotary to Reciprocation	0.176	333	0.53	20
Ratchet (cw)	0.086	65	1.3	1.9
Ratchet (ccw)	1.3	1089	1.2	100

Fig. 22. Experimental data (Sparcstation 10).

candidate designs, simply because it runs much faster. However, the local model is likely to be a better choice for an application in which, for example, unusual or innovative designs must be evaluated, since its simpler derivation and implementation make it more adaptable to new situations.

3. Long-term behavior prediction

The behavior of a machine with n state variables is a curve $s(t)$ in an n-dimensional state space. The function $s(t)$ is the solution to Eq. (1) (Section 2.2) for specific initial conditions, and $s(t)$ may be computed over any particular time period using numerical simulation, as described in Section 2.3. However, many machines exhibit repetitive behavior: $s(t)$ can be approximated as a simple function of t, at least for a discrete set of time points. Such an approximation is useful for predicting a machine's long-term behavior, since straight simulation over a very long time period can be quite expensive. The approximation also serves as a concise description of both the qualitative and quantitative aspects of the machine's predicted long-term behavior and thus may provide considerable insight.

The approximation used in my current implementation is a linear function of time

$$\hat{s}(t) = s(t_0) + (t - t_0)\Delta/D \tag{6}$$

where Δ and D are constant. Of course, a glance at Fig. 2 makes it clear that the behavior of something like an escapement is far from linear. However, by limiting the approximation to the time points at the start of each repetitive behavior pattern, a linear approximation becomes quite viable. Though the approximation only holds at these particular time points, intermediate values of $s(t)$ may be computed by running a detailed simulation over a time interval separating any two of the discrete time points where the approximation holds.

My program attempts to find such an approximation by using short, carefully controlled numerical simulations to find parameter values for Eq. (6). It then tests this hypothesized approximation by subjecting it to a series of tests in order to expose discrepancies between the approximation $\hat{s}(t)$ and the actual $s(t)$.

An alternative approach to predicting a machine's long-term behavior would be analysis using the mathematical theory of dynamical systems. For example, analyses of mechanical clocks may be found in [1,27,31]. These analyses use strong simplifying assumptions in order to reduce the number of state variables that must be considered to only two. In contrast, the less approximate behavioral model of a clock created by my program has six state variables, as described in Section 2. Reduction of the problem to two state variables is important for dynamical analysis, as it allows the use of the powerful Poincaré–Bendixson theorem, which provides a complete classification of attractors for two-dimensional systems and proves that chaotic behavior is impossible for these systems [7,26].

Thus long-term behavior prediction may be done either by using a detailed behavioral model and approximating the resulting behavior $s(t)$, or by approximating the behavioral model itself and then analyzing in detail the resulting approximate model. In this article

I take the first approach, though automating the approximation and dynamical analysis process needed for the second approach appears to be an interesting (and challenging) research problem. However, finding an simple approximation $\hat{s}(t)$ to the behavior seems likely to provide useful guidance in the task of generating an approximate behavioral model, so the two approaches are actually complementary rather than competitive. In addition, since the behavioral model used by my algorithms includes the complete mechanism geometry, the approximation $\hat{s}(t)$ based on that detailed behavioral model should provide a valuable check for an analysis based on a more approximate behavioral model which must necessarily abstract away most details of a mechanical device's geometry.

3.1. Finding behavioral regularities

My program attempts to approximate $s(t)$ at a series of discrete time points $t_0 + nD$, where t_0 is a time at which a particular repetitive behavior pattern begins, n is an integer, and D is the duration of the repetitive behavior patterns. The approximation currently used is the linear function of time in Eq. (6), which may be rewritten in terms of n

$$\hat{s}(t_0 + nD) = s(t_0) + n\boldsymbol{\Delta} \tag{7}$$

Though the approximation only holds at time points $t_0 + nD$ for integer n, intermediate values of $s(t)$ may be computed by running a detailed simulation with initial conditions $\hat{s}(t_0 + nD)$ for any n. For the simplest sort of periodic behavior, every component of the vector $\boldsymbol{\Delta}$ will be zero, and the approximation will be constant. However, a constant approximation won't work for something like the escapement, where Fig. 2 indicates that at least the entry of $\boldsymbol{\Delta}$ corresponding to the escape wheel position will be nonzero.

To find such an approximation, my program runs a numerical simulation to generate $s(t)$, searching for values of t_0, D, and $\boldsymbol{\Delta}$ which give an approximation matching $s(t)$ at the time points $t_0 + nD$. To constrain this potentially explosive search, the search process only examines states for which the function $e_k(s(t))$ has a minimum, where e_k is the kinetic energy of the mechanism. This restriction implies that the discrete time points $t_0 + nD$ will be kinetic energy minima. If the machine's behavior is in fact repetitive, it is also plausible to assume that each interval of length D will include the same number of kinetic energy minima.

My approximation algorithm focuses on kinetic energy minima because the presence of a kinetic energy minimum is a useful heuristic indicator of a boundary between qualitatively distinct regions of behavior, for example at a time when an oscillating part of a machine switches from moving in one direction to moving in another, or when a part of a machine receives a push or other energy boost so that its velocity stops decreasing and starts increasing. A change in the number of kinetic energy minima in a behavior pattern is thus a good indication of a significant qualitative change in behavior.

My program uses Algorithm 12 to try to find the approximation $\hat{s}(t)$ by filling in the fields of the data structure in Fig. 23 with values consistent with the stream of behavior data coming from the numerical simulator. The program simulates the behavior of the machine until it finds several kinetic energy minima. In order to avoid being misled by spurious local kinetic energy minima, the program only considers minima that are in the

1. Number of kinetic energy minima in an instance of the behavior pattern (M)
2. Duration of the behavior pattern (D)
3. Net change in each state variable over each pattern instance (Δ)
4. List of periodically superimposed processes (Fig. 25)

Fig. 23. Parameters for approximation $\hat{s}(t)$.

Algorithm 12. *Find a locally satisfactory behavioral hypothesis.*

[Default parameter values: $\mathcal{P}_{\text{confirm}} = 2$, $\mathcal{P}_{\text{range\_fraction}} = 10^{-5}$, $\mathcal{P}_{\text{ignore}} = 4$, $\mathcal{P}_{\text{t\_match}} = .01$, $\mathcal{P}_{\text{s\_v\_match}} = .02$]

While no behavioral hypothesis has been formed
    OR ⟨hypothesis belief level⟩ $< M * \mathcal{P}_{\text{confirm}}$
  Perform a simulation step
  Update range information
  If the previous state was a local kinetic energy minimum
      AND its energy level $< \mathcal{P}_{\text{range\_fraction}} *$ ⟨top of kinetic energy range⟩
      AND $\mathcal{P}_{\text{ignore}}$ kinetic energy minima have been ignored
  Then
    If there is no hypothesis
    Then hypothesize a behavior pattern:
      $M \leftarrow 1$
      $D \leftarrow$ difference in time between the current kinetic energy minimum and the previous one
      $\Delta \leftarrow$ differences in the values of the state variables between the current kinetic energy minimum and the previous one
    Else the current hypothesis remains valid if and only if:
      D matches the time elapsed since the Mth previous kinetic energy minimum to within $\mathcal{P}_{\text{t\_match}}$
      AND Δ matches the changes in the values of the state variables since the Mth previous kinetic energy minimum to within $\mathcal{P}_{\text{s\_v\_match}}$
      If the current hypothesis fails then form a new hypothesis:
        ⟨hypothesis belief level⟩ $\leftarrow 0$
        $M \leftarrow M + 1$
        $D \leftarrow$ difference in time between the current kinetic energy minimum and the Mth previous one
        $\Delta \leftarrow$ differences in the values of the state variables between the current kinetic energy minimum and the Mth previous one
      Else increment ⟨hypothesis belief level⟩

bottom $\mathcal{P}_{\text{range\_fraction}}$ of the range of kinetic energies encountered during the simulation. The first $\mathcal{P}_{\text{ignore}}$ minima are ignored so that the program will not be misled by transient startup phenomena. Machines designed for regular behavior must incorporate damping or negative feedback to rapidly eliminate behavioral irregularities, so $\mathcal{P}_{\text{ignore}}$ can safely be set to a small value.

The program searches for a viable approximation by successively forming each possible hypothesis which is consistent with the currently available data. After each hypothesis is formed it is tested against the results of further simulation. Though the program requires that new data match the predictions of the hypothesis fairly closely, it does not require exact matches because of transients and other noise in the numerical data. The match parameters can be adjusted to "tune" the behaviors the program will recognize. For example, identical patterns with very different time durations could be detected by giving $\mathcal{P}_{\text{t\_match}}$ a large value.

This algorithm is based on the following simple but powerful ideas:

(1) Given a set of simulation data including exactly m kinetic energy minima, if M in Fig. 23 is assigned the value m, then there is clearly a unique assignment of D and Δ which is consistent with the data.

(2) Given a set of simulation data including more than m kinetic energy minima, either the single m-minima hypothesis which is consistent with the first m minima will also be consistent with the rest of the data, or else no m-minima hypothesis can be consistent with all of the data, which implies that:

(3) Once the program has found new data not consistent with a previously formed m-minima hypothesis (which was consistent with some particular sequence of m minima), it need never again consider any hypothesis with m kinetic energy minima.

Thus the program can iterate through the possible behavioral hypotheses, testing each in turn.

Note that the program looks for regularities in the behavior of the state variables (like that shown in Fig. 2), not in the behavior of the kinetic energy. Kinetic energy is simply used as a guide to tell the program when to look at the state variables—it might be considered a measure of the "interestingness" of a particular state of the machine. Focusing the search on the most significant points in the system's behavior reduces the chance of incorrect matches, and also makes Algorithm 12 quite efficient. If a machine has a behavior pattern with M kinetic energy minima, and the behavioral simulator takes no more than S steps during each instance of the behavior pattern, then Algorithm 12 will terminate after no more than $S(\mathcal{P}_{\text{ignore}} + M(\mathcal{P}_{\text{confirm}} + 1))$ iterations of its main loop.

A potential drawback in using kinetic energy minima as a filter when analyzing a machine's behavior is that some interesting behaviors could be filtered out. For example, any machine having no energy input, no potential energy storage mechanism, and no friction or other dissipation or energy output will clearly have absolutely constant kinetic energy at all times, even though it may have complex and interesting motions. However, such machines are not common; even a machine as simple as a frictionless pendulum fails to fully meet these criteria. Transformations between different forms of energy are central to the purpose of most machines.

Algorithm 13. *Test for gradual effects.*

Let $n_2 = \left\lfloor \frac{1}{2} \frac{\langle \text{initial total energy} \rangle}{\langle \text{change in total energy per pattern instance} \rangle} \right\rfloor$

Apply Algorithm 12 with initial conditions $\hat{s}(t_0 + n_2 D)$
If the new hypothesis does not match the initial locally satisfactory hypothesis
  Then FAILURE
Else let $s(t_0 + n_b D)$ be the last state reached in generating the initial locally satisfactory hypothesis
  $s(t_0 + n_a D)$ be the last state reached in generating the new locally satisfactory hypothesis
  If $s(t_0 + n_a D) \approx s(t_0) + n_a \Delta$
    Then SUCCESS
  Else reset Δ to be $\frac{(s(t_0 + n_a D) - s(t_0 + n_b D))}{(n_a - n_b)}$

  And test the new hypothesis at t_0 using the validity test in Algorithm 12

3.2. *Hypothesis failures*

Algorithm 12 finds a "locally satisfactory" behavioral hypothesis: an approximation $\hat{s}(t)$ which is consistent with the initial simulation data. If $\hat{s}(t_0 + nD)$ remains close to $s(t_0 + nD)$ for all n then the hypothesis is also globally satisfactory; otherwise the hypothesis must eventually fail. In this article I consider two classes of hypothesis failure:

- the distance between $\hat{s}(t_0 + nD)$ and $s(t_0 + nD)$ gradually grows as n increases, eventually becoming large
- A sudden transition in the machine's behavior $s(t)$ occurs at some time t_a, so that for $n > (t_a - t_0)/D$, the behavior $\hat{s}(t_0 + nD)$ predicted by the approximation is far from the actual behavior $s(t_0 + nD)$

My program performs a variety of heuristic tests in order to try to detect these hypothesis failures. If a hypothesis passes all of the tests it is declared globally satisfactory and therefore useful for long-term behavior prediction. If the hypothesis fails a test, it is modified if possible or otherwise rejected.

3.2.1. *Gradual effects*

Imprecisions in the components of Δ may cause the approximation

$$\hat{s}(t_0 + nD) = s(t_0) + n\Delta \tag{8}$$

to gradually deviate from the actual behavior $s(t_0 + nD)$. At a time $t_0 + n_2 D$ when $\hat{s}(t_0 + n_2 D)$ deviates significantly from $s(t_0 + n_2 D)$, $\hat{s}(t_0 + n_2 D)$ will often represent an unstable state of the machine, and my program uses Algorithm 13 to look for such instabilities by running a short simulation with initial conditions $\hat{s}(t_0 + n_2 D)$. If a

deviation is detected in this way, Algorithm 13 attempts to modify Δ to improve the approximation $\hat{s}(t)$.

The escapement mechanism provides an example of such a gradual effect. As the clock runs, the decreasing tension in the mainspring results in a very slow decrease in the amplitude of the oscillations of the balance. However, this decrease is smaller than the background noise in the simulation data, so it is not detected when Algorithm 12 searches for an initial locally valid $\hat{s}(t)$, and the initial Δ indicates zero change for the balance between the starts of successive behavior pattern repetitions. However, when Algorithm 13 restarts the simulation with initial conditions $\hat{s}(t_0 + n_2 D)$, the gradual change in balance amplitude has grown far larger than background noise, and Algorithm 13 can easily correct Δ to include a small but nonzero change in balance position over each behavioral pattern.

3.2.2. *Sudden transitions*

Figs. 26–29 show mechanisms for which hypotheses fail due to sudden transitions in the machine's behavior $s(t)$. For example the "time bomb" in Fig. 29 will behave like a normal clock for quite a while and then suddenly stop because a protrusion on one of the wheels will run into a protrusion on the frame. This sort of hypothesis failure is quite different from the failures discussed in the previous section because Algorithm 13 is unlikely to detect the sudden transition. Of course Algorithm 13 would detect the sudden transition if $\hat{s}(t_0 + n_2 D)$ happened to be a state just before the wheel collided with the frame, but if $\hat{s}(t_0 + n_2 D)$ were instead the (actually unreachable) state with the wheel on the other side of the protrusion, then Algorithm 13 would have no way to detect that the actual behavior $s(t)$ had deviated greatly from the predicted behavior $\hat{s}(t)$.

In the mechanical devices we are considering, sudden transitions are changes in the patterns of contact between parts in the mechanism. I divide these changes into two categories:

(1) changes in the pattern of contact between two parts which regularly make contact;

(2) new contacts between parts not previously in contact.

The escapement mechanisms with modified escape wheels in Figs. 27 and 28 are examples of the first category. The escape wheels in these two mechanisms are modified so that the pattern of contact between the escape wheel and the lever will change after a while, possible violating a successful local hypothesis. The chiming clock (Fig. 26) and the "time bomb" (Fig. 29) are examples of the second category.

My program uses Algorithm 14 to make sure changes in the pattern of contact between two parts which regularly make contact will not violate the hypothesis. Moving parts which return to the same state at the end of each behavioral pattern cannot change their pattern of contact, so the algorithm only considers elementary kinematic subsystems with hypothesized position changes. The algorithm is limited to elementary kinematic subsystems which are revolute pairs, since the elements of a prismatic pair with a net position change per pattern would soon separate. In the case of a rotating part, Algorithm 14 either uses symmetry considerations to rule out the possibility of sudden transitions, or it raises the number of confirmations required to make the hypothesis

Algorithm 14. *Check for symmetric regular contacts.*
For each elementary kinematic subsystem *EKS*
If *EKS* has a hypothesized position change
AND *EKS* is a revolute pair
AND *EKS* has interacted with another elementary kinematic subsystem *EKS2*
Then let *M* be $2\pi/\langle$the hypothesized position change for *EKS*$\rangle$
For each primitive *PR* in *EKS* which could interact with *EKS2*
If *PR* is not part of a union having *M*-radial or rotational symmetry
Then raise the number of required confirmations to *M*

locally satisfactory so that the motion of the part will be simulated through a full revolution.

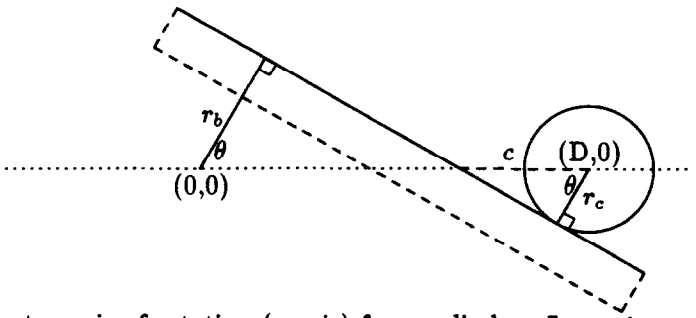
The program applies Algorithm 14 after Algorithm 12 has found a locally satisfactory hypothesis. If Algorithm 14 raises the number of required confirmations, Algorithm 12 is applied again until the new requirement has been met.

My program determines whether a primitive solid is part of a symmetric union by checking whether the primitive was defined using the SYMMETRIC_UNION macro, a syntactic extension I added to the PADL-2 solid modeling language. This macro takes five arguments: a primitive solid, an axis of symmetry, a type (ROTATIONAL or TRANSLATIONAL), a repetition count, and a range. The preprocessor expands this into an explicit union, in the PADL-2 language, having the specified properties, and also marks each resulting primitive as having come from a particular symmetric union. Algorithm 14 checks each primitive solid *PR* to see if it is part of an appropriate symmetric union or has the appropriate symmetry itself, as a properly positioned cylinder might.

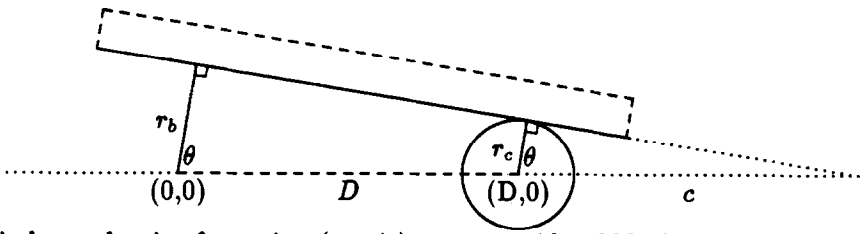
3.2.3. New contacts between parts not previously in contact

New contacts between parts not previously in contact may be classified into a number of categories, only some of which are handled by my current program. The two parts coming in contact for the first time may both be moving, or one may not have moved before they come into contact. The moving part(s) may be rotating or translating. The primitive solids which come into contact may be either blocks or cylinders. My current implementation predicts contacts between a block and a cylinder. One of them must be motionless before they come into contact, and the other must rotate about a fixed axis. The algorithms operate in the same "$2\frac{1}{2}$-dimensional" world described earlier in Section 2.2.1. Extension of the algorithms I will describe to cover all cases of a moving part running into a motionless part would be straightforward. Predicting contacts between two simultaneously moving parts is considerably more difficult; a general approach would probably require the use of search techniques.

Before presenting the algorithms my program uses to predict contacts between a block and a cylinder, I will derive some geometrical preliminaries. Fig. 24 shows a block and a cylinder, capable only of rotation about a common fixed axis of rotation.



a) Block separates axis of rotation (z-axis) from cylinder: $\Psi = -1$



b) Cylinder and axis of rotation (z-axis) on same side of block: $\Psi = 1$

Fig. 24. The two ways a cylinder can touch within a face of a block.

(The algorithms assume only one of them moves, but this issue is irrelevant from the geometrical point of view.) The following symbols are used in the figure:

- D the distance from the axis of the cylinder to the axis of rotation,
- c the distance from the axis of the cylinder to the intersection between the plane of the two axes and the plane containing the face of the block,
- θ the angle between the plane perpendicular to the face of the block which contains the axis of rotation and the plane containing the two axes,
- r_b the distance from the axis of rotation to the face in contact with the cylinder,
- r_c the radius of the cylinder,
- Ψ 1 if the axes are separated, and -1 if they are not.

By the properties of similar triangles,

$$\frac{c}{r_c} = \frac{D + \Psi c}{r_b}, \tag{9}$$

so that

$$c = \frac{D}{r_b/r_c - \Psi}. \tag{10}$$

Therefore

$$\cos\theta = \frac{r_c}{c} = \frac{r_b - \Psi r_c}{D}. \tag{11}$$

PSP:

1. Period (time between each start of the superimposed process)
2. Duration (how long the process runs)
3. Time at which process first starts
4. Vector of truth values specifying involved state variables
5. Net change in each involved state variable over a process

Fig. 25. Periodically superimposed process (PSP) data structure.

It is also possible that a cylinder may not touch a block within a face but rather at the edge of a face (i.e., in Fig. 24, at one of the blocks' corners). In this case the law of cosines requires that

$$r_c^2 = r_e^2 + D^2 - 2r_e D\cos\theta_e, \tag{12}$$

where θ_e is the angular polar coordinate of the "corner" of the block when it is in contact with the cylinder, and r_e is the corner's radial polar coordinate, which is constant since the block rotates rigidly. Therefore

$$\cos\theta_e = \frac{r_e^2 + D^2 - r_c^2}{2r_e D}. \tag{13}$$

In either case there are two possible solutions for θ, due to the fact that the figures may be reflected across the horizontal axis. The algorithms presented below choose the appropriate solution.

When a new contact occurs, several things may happen. If one of the parts is fixed, then the hypothesis will fail completely at the time of contact. Simulation can then be used to determine the subsequent behavior of the mechanism; typically it will quickly come to a halt. On the other hand, if the part which was not moving prior to the contact is capable of motion, a new behavior pattern may emerge.

Many mechanisms are multiply periodic; different sorts of regular behavior occur at very different time scales. For example, a chiming clock like that in Fig. 26 will have an escapement with a regular behavior pattern that might be repeated several times a second, and a chiming mechanism whose pattern might only be repeated once an hour. Perhaps the commonest case of multiple periodicity is what I call a *periodically superimposed process* (PSP): an additional behavior pattern which is regularly superimposed on the basic behavior pattern without disturbing it. My program represents periodically superimposed processes with the PSP data structure (Fig. 25).

Because a superimposed process does not disturb the basic behavior pattern, it may involve only state variables which don't change during the basic pattern. I call these state variables *static*. The static state variables are identified during the simulation needed to generate the original local hypothesis.

My program uses Algorithm 15 to analyze behavior resulting from a new contact. The motion envelope of a part is the volume it sweeps out as it moves. If two parts have intersecting motion envelopes but have never been in contact, their first point of contact is computed using Algorithm 16. The parts can only come in contact if the behavioral

Algorithm 15. *Analyze behavior resulting from a new contact.*

1. Use Algorithm 16 to determine how many behavioral pattern instances will pass before the first nonhypothesized contact
2. Use Algorithm 13 with initial conditions given by $\hat{s}(t)$ for t halfway to new contact to check for gradual effects of steadily changing state variables, and modify hypothesis if necessary
3. If neither part is fixed, restart behavior simulator with initial conditions given by $\hat{s}(t)$ for t just before new contact. While simulating,
   (a) Monitor behavior of nonstatic state variables to make sure none of them violate the original hypothesis
   (b) Monitor behavior of static state variables to gather data needed to fill in PSP data structure
4. If the superimposed process does not violate the original hypothesis, apply Algorithm 13 in default mode to for gradual effects when half the total energy is gone, and modify hypothesis if necessary

Algorithm 16. *Look for nonhypothesized contacts.*

```
For each INTERACTOR data structure (Fig. 9)
    If the two parts have never been in contact
      If both parts move then print a warning message
      Else if the moving part has a nonzero position change per behavioral pattern
            For each pair of potentially interacting primitive solids
                If the one solid is a block and the other is a cylinder
                  For each of the four edges of the block's 2D projection
                      Use Algorithm 17 to find out when the edge will hit the
                      cylinder
                Else print a warning message
            Divide the distance the part can move by its hypothesized displacement
            per basic behavioral pattern instance
```

hypothesis for the mechanism predicts a net change in one of the parts' positions over each behavioral pattern instance. Since there may be several nonhypothesized contacts, the first is determined by dividing the distance each of the moving parts can move freely before the contact by the part's hypothesized displacement per behavioral pattern. Thus the number of behavioral pattern instances without nonhypothesized contacts can be determined. If one of the parts is fixed (part of the frame), then my program determines that the valid region for the original hypothesis ends at the new contact; otherwise, the program attempts to identify a new periodic process that will be superimposed on the originally hypothesized behavior pattern.

Algorithm 16 uses Algorithm 17 to find the first contact between one of the four edges of the block's two-dimensional projection and the circle which is the cylinder's

Algorithm 17. *Find first contact between a block's edge and a cylinder.*
For each of the two cases $\{\Psi = -1 \ ; \ \Psi = 1\}$
  If $|\Psi r_c + r_b| \geqslant D$
    For each of the two cases $\{S = -1 \ ; \ S = 1\}$
      Let $\theta = S \arccos\left(\dfrac{r_b - \Psi r_c}{D}\right)$
      If the contact point $(D + \Psi r_c \cos\theta, \Psi r_c \sin\theta)$ lies between the endpoints of the edge of the block
        Then record this θ as a possible contact
      Else if the more distant endpoint is nearer than the contact point, but its distance r_e from the axis of rotation is greater than $D - r_c$
        OR if the nearer endpoint is farther away than the contact point, but its distance r_e from the axis is less than $D + r_c$
        Then $\theta_e = S \arccos\left(\dfrac{r_e^2 + D^2 - r_c^2}{2 r_e D}\right)$ is a possible contact
The first contact is the nearest contact in the direction of motion of the moving solid

projection. Each of the two contact cases shown in Fig. 24 can be reflected across the horizontal axis, giving four possibilities. In some of these cases the body of the block will be inside the cylinder and in others it will be outside. However, since the final output of the algorithm is the nearest contact, the physically unrealizable cases will be automatically eliminated since a physically possible contact must necessarily happen first. If appropriate, the algorithm also checks the four ways an endpoint of the edge can touch the cylinder.

3.3. Examples

The algorithms in this section have all been implemented and tested on a number of examples. The behavior pattern of the escapement mechanism in Fig. 1 has two kinetic energy minima, one at each extreme position of the balance. The initial hypothesis formed by Algorithm 12 is that each behavior pattern instance has only one kinetic energy minimum, and that all three moving parts have a net position change per pattern. When Algorithm 12 tests this behavioral hypothesis at the next minimum, it finds that the net displacements of the balance and lever are the negatives of the hypothesized changes. The initial behavioral hypothesis is then rejected, and a new hypothesis is formed in which the behavior pattern has two kinetic energy minima, and neither the balance nor the lever has a net position change per pattern. This hypothesis is then confirmed over the next $\mathcal{P}_{\text{confirm}}$ pattern instances, and finally accepted as locally valid.

After finding a locally acceptable hypothesis, the program applies Algorithm 13 as the first check for hypothesis failure. The simulation is restarted at a time $t_0 + n_2 D$ when approximately half of the clock's original energy (stored in the mainspring) has been dissipated. Algorithm 13 reapplies Algorithm 12 with this new initial condition, and arrives at the same approximation $\hat{s}(t)$ that it had before. However, when the

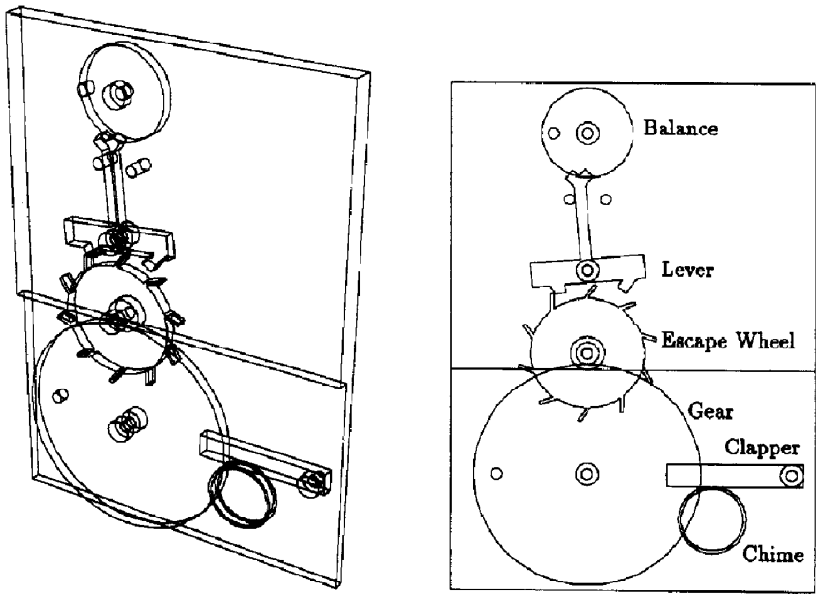


Fig. 26. Chiming clock.

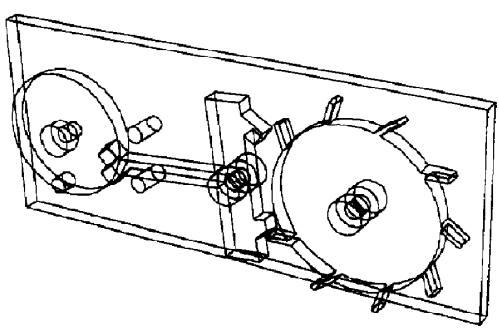


Fig. 27. Escapement mechanism with missing tooth.

state $s(t_0 + n_a D)$ was compared to the prediction $s(t_0) + n_a \Delta$, Algorithm 13 found a significant deviation because the amplitude of the oscillations of the balance had become considerably smaller, due to the decrease in tension in the mainspring. The program then executes the final portion of Algorithm 13, changing Δ to include the small change in balance amplitude over each behavior pattern repetition. The new approximation $\hat{s}(t)$ is then tested and found to work locally both at t_0 and $t_0 + n_2 D$. No sudden transitions are found for this example.

Fig. 26 shows a clock which chimes at regular intervals (e.g. every hour). A small gear on the escape wheel drives a large gear, which moves relatively slowly. The clapper

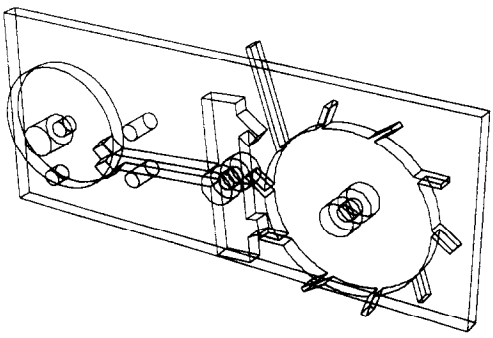


Fig. 28. Escapement mechanism with extra-large tooth.

is normally pressed against the chime by a spring, but the protrusion on the large gear pushes it away from the chime and then releases it so that it makes a chiming sound when it hits the chime again. In this example, the program proceeds to find a local hypothesis just as it did in the previous example; the only significant difference is that there are five elementary kinematic subsystems instead of three. Since the clock will have been running for quite a while before it first chimes, the initial hypothesis will not take the chiming into account and will therefore be an incomplete behavior prediction. Using the algorithms in Section 3.2.3, the program determines that a nonhypothesized contact will occur between the large gear and the clapper. It then restarts the simulation with initial conditions given by $\hat{s}(t)$ halfway between t_0 and this unexpected collision in order to check for gradual effects, but no hypothesis modification is needed. The program then restarts the simulation with initial conditions given by $\hat{s}(t)$ just before the collision in order to determine the results of the nonhypothesized contact, and it identifies a periodic process which is superimposed on the basic hypothesis without affecting it. The program applies Algorithm 13 as usual at a time when about half of the clock's energy is gone to check again for gradual effects, and this time it does need to modify the approximation $\hat{s}(t)$ to reflect a decrease in the amplitude of oscillation of the balance resulting from decreased tension in the mainspring.

Fig. 27 shows an escapement mechanism which is missing one of the teeth on the escape wheel, and Fig. 28 shows an escapement mechanism in which one of the teeth on the escape wheel is exceptionally large. For both of these mechanisms, my program starts by finding the same hypothesis that it finds for the standard escapement. But in these examples, the escape wheel is not symmetric, so when the program attempts to locally confirm the hypothesis it must simulate through a full rotation of the escape wheel.

In the case of the mechanism with the extra-large tooth in Fig. 28, before the escape wheel has completed a full revolution, the large tooth contacts the lever and prevents the escape wheel from advancing further, so the initially successful hypothesis fails. My program then attempts to form other hypotheses, but none of them are confirmed and eventually it gives up.

In the case of the mechanism with the missing tooth in Fig. 27, the initially successful

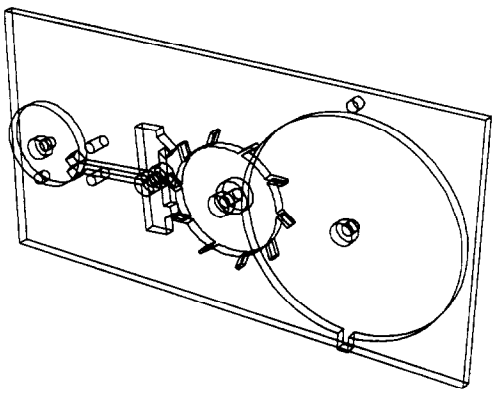


Fig. 29. The "time bomb".

hypothesis also fails before the escape wheel completes a full revolution. However, when the program starts forming new hypotheses, it eventually finds one with 20 kinetic energy minima in which the displacement of the escape wheel position state variable is one full revolution, and this new hypothesis turns out to be completely successful.

Fig. 29 shows a "time bomb", a clock which runs normally for a long time and then suddenly exhibits anomalous behavior by stopping when a protrusion on one of the wheels runs into a protrusion on the frame. In this example, Algorithm 12 finds a local hypothesis just as it did in the previous cases. The algorithms in Section 3.2.3 determine that an unexpected collision will occur. Algorithm 13 is used to check for gradual deviations of $\hat{s}(t)$ halfway between t_0 and the collision. In this example, the amplitude of the balance oscillation has not changed significantly because the tension in the mainspring hasn't changed much, so the original hypothesis is still considered globally consistent over the interval ending when the unexpected collision occurs.

4. Related work

A number of mechanical device simulators are commercially available, such as ADAMS [6,29], DADS [22], and others (see surveys in [8,21,30,32]). Though these simulators include powerful algorithms for forming and solving the equations of motions for a wide variety of mechanisms, they are incapable of using information about the shape of a machine's parts, so the user of the simulator inherits the following modeling tasks:

- For each pair of parts in permanent contact, completely describe the resulting geometric constraint.
- For each pair of parts which may intermittently come in contact, supply a subroutine which for any positions of the two parts will determine whether or not they are in contact or interpenetrating.

These tasks are both done automatically by my program.

These tasks are also addressed by Joskowicz and Sacks [25]. They handle permanent contacts by transforming all surface-to-surface contacts in the initial configuration into potential axes and planes of motion, which are then intersected to compute axes of kinematic pairs. They handle intermittent contacts by computing reachable regions of pairwise configuration spaces, which are two-dimensional. This approach of computing the entire set of possible contacts in advance is quite different from my approach of locally computing the actual contacts which occur during a simulation. In [37] they supplement their kinematic analysis by adding a kinematic simulator which allows them to simulate the behavior of many mechanisms; however, kinematic simulation is not adequate to predict the behavior of a mechanism like an escapement, which requires the sort of detailed dynamic analysis I have presented in this article.

Nakamura and Nakajima [28] describe a kinematic pair recognition program. For each pair of parts in a mechanism represented using a feature description language, the program identifies a set of matching features and then looks for the set in a list of the sets of matching features found in some instantiations of certain common kinematic pairs.

The handling of intermittent contacts in mechanism simulation been addressed by Cremer [4], Gilmore and Cipra [17] and Conti et al. [3]. Like the commercial simulators, these researchers require geometric constraints resulting from permanent contacts to be specified as input to their programs. They all represent contacts with geometric constraints, as I do in Appendix A; however, each then simulates using systems of constrained equations rather than using the constraints as I do to form an unconstrained system of equations having fewer degrees of freedom. The "soft contact" sort of approach to intermittent contacts that I describe in Section 2 has also been used by Cundall [5] and Goyal et al. [19], though in both cases only for sets of unconstrained rigid bodies, not for mechanical devices.

None of the above research cited above addresses the problem of predicting a machine's long-term behavior based on numerical simulation output. Long-term behavior prediction is one of the goals of the mathematical theory of dynamical systems, and, for example, analyses of very abstract models of mechanical clocks may be found in [1,27,31]; Section 3 discusses this approach. Weld [39] describes a program which detects repeating cycles of processes and produces an aggregate description of the repetitive behavior. The behavior of the mechanical devices described in this article does not easily decompose into a series of discrete processes, so Weld's technique does not appear usable in this context.

Forbus et al. [11] describe a system which does qualitative kinematic analysis and simulation for a clock. Their simulation suffers from a qualitative model's inherent liability to make ambiguous behavior predictions, and thus predicts that the clock may run, but not whether it will run and if so, for how long. Furthermore the simulation, being qualitative, cannot predict what the period of the clock will be or whether it will be constant, which would seem to be the defining characteristic of a clock.

Yip [41] and Sacks [36] describe intelligent controllers for numerical simulations of physical systems. Their systems require behavioral models (equations) as input; they do not address the issue of model creation. Yip's program can automatically plan, execute, and interpret numerical experiments concerning Hamiltonian systems with two

degrees of freedom. Sacks' program can automatically form a detailed analysis of one-parameter planar ordinary differential equations. Both programs are quite powerful in their domains of expertise, though neither could handle a sixth-order system like the escapement mechanism I present in this article.

Forbus and Falkenhainer [10] describe an intelligent controller for numerical simulations based on Qualitative Process Theory [9]. Their work emphasizes automatically generating a system's equations, which may change over time, from a physical model. Their input model is, however, at a considerably higher level than the model of raw physical structure my program uses as input, which would prevent their program from being able to generate a numerical simulation of a device like a clock in which contacts between parts appear and disappear dynamically.

This article is mainly based on my dissertation, [15]. Less detailed descriptions of some of the work described in this article appear in [13, 14, 16].

5. Limitations and future work

Combining my modeling algorithms from Section 2 with the capabilities of the commercial mechanical device simulators described in Section 4 would result in a system having a considerably wider range of applicability than either of its components. My dynamics modeling algorithms in Section 2.2 are presently limited to fixed-axis mechanisms, while the commercial simulators can't make use of information about the shape of a machine's parts, thus imposing a considerable modeling burden on the user, especially for mechanisms with intermittent contacts, as described in Section 4. The commercial simulators can handle complex movable-axis mechanisms, and my algorithms can automatically identify both permanent and intermittent contacts, so the two approaches are complementary and remedy each other's deficiencies.

The principal limitation of the long-term behavior prediction algorithms in Section 3 is the form of the approximation function given in Eq. (7). The set of machines whose long-term behavior can be predicted is necessarily limited to those whose behavior (at a set of discrete time points) can be adequately approximated by a function of this form. This limitation is also a strength: a constrained form for the approximation function makes possible the use of more powerful behavior recognition algorithms. An important direction for future work in this area is the investigation of how the set of possible approximations might usefully be expanded without precluding tractable behavior recognition. The question of what generalizations might be useful is probably best answered by pursuing the research in the context of particular tasks like diagnosis of malfunctioning machines or design of new machines.

6. Conclusion

Predicting a machine's behavior is a basic capability for many tasks requiring reasoning about machines, such as diagnosis of malfunctioning machines, design of new

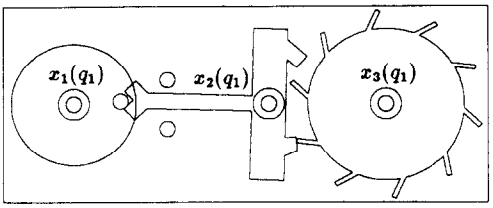


Fig. A.1. The position variables x_i of the kinematic pairs are functions of the position variables q_i of the kinematic subsystems.

machines, and redesign of existing machines. I have described a working program with the following capabilities:

- automated creation of behavioral models of machines directly from models of their raw physical structure;
- intelligent control of the computational experiments needed to reveal a machine's long-term behavior.

The algorithms I present for solving these problems allow the automated prediction of a machine's behavior over both short and long time periods, and the program's final output is a concise qualitative/quantitative description of the machine's expected long-term behavior.

Appendix A. Global model for contact forces

Contact forces between parts may result either from sudden collisions or from steady pushing of one part by another. The model for contact forces presented in Section 2.2.1 treats collisions and pushing in a single uniform way, but in this appendix the two must be dealt with separately. Pushing is handled by forming temporary kinematic pairs, and collisions are "special case" momentum transfers which take place as new temporary kinematic pairs are formed. Collisions between moving parts are idealized as completely inelastic. This assumption is plausible for several reasons of which the most important concern the damping effects of lubrication between colliding parts and between the elements of permanent kinematic pairs [2].

A.1. Theory

A.1.1. Equations of motion

Consider a mechanism in which each moving part forms a permanent kinematic pair with the frame. Each of these kinematic pairs has an associated position variable x_i. At any particular time the contacts between the moving parts will group them into m kinematic subsystems each having a single degree of freedom. For example, in Fig. A.1 the three moving parts have temporarily joined together into a single kinematic subsystem which has a total of one degree of freedom.

Let q_j be the position state variable for the jth kinematic subsystem. If the kinematic subsystem has only one moving part, q_j will be the position variable for the kinematic pair that part forms with the frame, and if the subsystem has several moving parts, then q_j will typically be the position variable of one of the parts; by the definition of kinematic subsystem, the positions of the other moving parts will be determined by q_j.

Let L be the Lagrangian of the mechanism, the difference between its kinetic energy and its potential energy. The behavior of the mechanism can be modeled by m differential equations of the form

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial L}{\partial \dot{q}_j}\right) - \frac{\partial L}{\partial q_j} = G_j \tag{A.1}$$

where G_j is the jth component of the generalized force. (This is the *Lagrangian* formulation of classical mechanics, which is equivalent to Newton's formulation but more convenient for some problems [18].)

Using the definitions from Section 2.2.1, the Lagrangian of the mechanism is

$$L = \frac{1}{2}\sum_i \left(I_i \dot{x}_i^2 - k_i x_i^2\right), \tag{A.2}$$

and the components of the generalized force are [18, p. 19]

$$G_j = \sum_i -h_i \dot{x}_i \frac{\partial x_i}{\partial q_j} \tag{A.3}$$

Substituting into Eq. (A.1) and simplifying gives

$$\sum_i \frac{\mathrm{d}x_i}{\mathrm{d}q_j}\left(I_i\left(\frac{\mathrm{d}x_i}{\mathrm{d}q_j}\ddot{q}_j + \frac{\mathrm{d}^2 x_i}{\mathrm{d}q_j^2}\dot{q}_j^2\right) + k_i x_i + h_i \frac{\mathrm{d}x_i}{\mathrm{d}q_j}\dot{q}_j\right) = 0 \tag{A.4}$$

which may be rearranged to give

$$\ddot{q}_j = \frac{-\sum_i \frac{\mathrm{d}x_i}{\mathrm{d}q_j}\left(I_i \frac{\mathrm{d}^2 x_i}{\mathrm{d}q_j^2}\dot{q}_j^2 + k_i x_i + h_i \frac{\mathrm{d}x_i}{\mathrm{d}q_j}\dot{q}_j\right)}{\sum_i I_i \left(\frac{\mathrm{d}x_i}{\mathrm{d}q_j}\right)^2} \tag{A.5}$$

which is the desired input form for a numerical simulator. See [15] for details of the above derivation. To turn this equation into a useful behavioral model of a mechanism requires algorithms for computing x_i, $\mathrm{d}x_i/\mathrm{d}q_j$, and $\mathrm{d}^2 x_i/\mathrm{d}q_j^2$ as functions of q_j. These algorithms are presented in Section A.2, and are based on the geometrical theory described in Section A.1.3.

A.1.2. Collisions

At any particular time the elementary subsystems in a mechanism will be grouped into m kinematic subsystems each having a single degree of freedom, the state of the mechanism will be specified by $2m$ state variables q_j and $\dot{q}_j$, and the behavior of the

mechanism will be determined by m equations, an instance of Eq. (A.5) for each of the m subsystems.

When the motion of the mechanism causes two of the m kinematic subsystems to come into contact, then these two kinematic subsystems may join together to form a new subsystem having a single degree of freedom, and both the set of state variables and the equations of motion will change. For example, in Fig. 1(b), the balance, which had been moving independently, comes in contact with the lever, and they begin moving together.

A new contact begins with a collision. In the rigid-body model considered in this section, collisions are assumed to take place within an infinitesimally short time interval during which the velocity state variables may change their values, but the position state variables remain constant [35,40]. The force $\boldsymbol{F}$ between the colliding subsystems is an *impulsive* force because the integral $\int_t^{t+\Delta t} \boldsymbol{F} \mathrm{d}t$ converges to a finite nonzero value $\hat{\boldsymbol{F}}$ as the time interval Δt approaches zero. During the collision Eq. (A.4), the equation of motion for a kinematic subsystem, has an additional generalized force term and becomes

$$\sum_i \frac{\mathrm{d}x_i}{\mathrm{d}q_j}\left(I_i\left(\frac{\mathrm{d}x_i}{\mathrm{d}q_j}\ddot{q}_j + \frac{\mathrm{d}^2x_i}{\mathrm{d}q_j^2}\dot{q}_j^2\right) + k_i x_i + h_i\frac{\mathrm{d}x_i}{\mathrm{d}q_j}\dot{q}_j\right) = F\frac{\mathrm{d}x_F}{\mathrm{d}q_j} \tag{A.6}$$

where F is the magnitude of force $\boldsymbol{F}$ and x_F measures distance in the direction of $\boldsymbol{F}$. Integrating this equation over the time of the collision yields

$$(\dot{q}_j^{\text{after}} - \dot{q}_j^{\text{before}})\sum_i I_i\left(\frac{\mathrm{d}x_i}{\mathrm{d}q_j}\right)^2 = \hat{F}\frac{\mathrm{d}x_F}{\mathrm{d}q_j}, \tag{A.7}$$

where $\hat{F}$ is the magnitude of impulse $\hat{\boldsymbol{F}}$, using the assumption that collisions take place within an infinitesimally short time interval during which the position state variables remain constant.

A.1.3. Geometry

Consider two moving parts in a mechanism which each form kinematic pairs with the fixed frame. If these parts are temporarily in contact, it is often necessary to compute the position state variable of one of the pairs from that of the other in order to make use of the equations derived in the previous sections. My current implementation is "$2\frac{1}{2}$-dimensional": it works by analyzing two-dimensional projections of three-dimensional situations.

In this section I will discuss the case of temporary contact between the moving parts of two revolute pairs with parallel axes. I will start with the case in which a corner of a block which is a component of one of the moving parts touches an edge of a block which is a component of the other moving part. In the escapement mechanism discussed in earlier sections, the contacts between the lever and escape wheel are of this type.

I will refer to the first moving part as part C (for corner) and to the kinematic pair which it forms with the frame as pair C. The other part is part E (for edge) which is an element of pair E. In the following discussion I will use a coordinate system in which the axis of rotation of revolute pair E coincides with the z-axis, and the axis of rotation

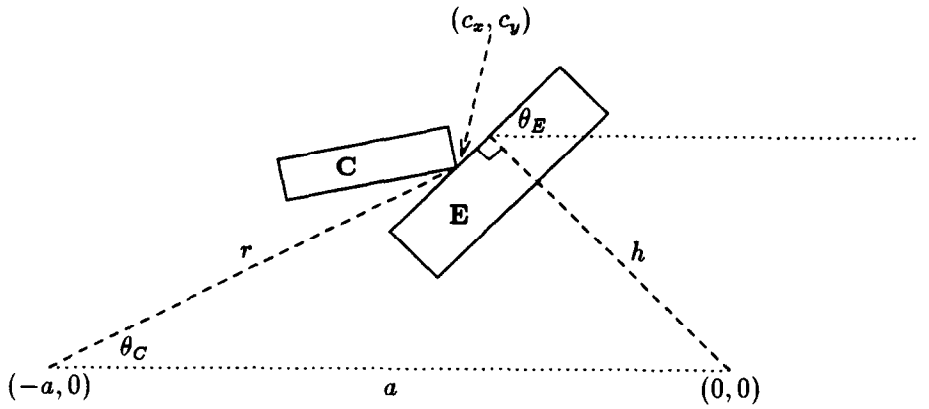


Fig. A.2. A corner in contact with an edge.

of revolute pair C is parallel to the z-axis and intersects the negative x-axis. The contact between the parts is a line segment parallel to the z-axis. I will discuss the situation as projected on the X–Y plane, and therefore I will refer to the contact as a point, with coordinates (c_x, c_y). See Fig. A.2. I define the following symbols:

- a the distance between the axes of the revolute pairs,
- r the distance from the axis of rotation of C to the point of contact (c_x, c_y),
- h the shortest distance between the origin and the projection of the edge of part E involved in the contact,
- θ_C the angular orientation of the point of contact (c_x, c_y) relative to the axis of rotation of C,
- θ_E the angle between the edge of E and the horizontal.

Using plane geometry and trigonometry, we can then derive the following relations between the positions of the two moving parts (see [15] for details):

$$c_y = r \sin \theta_C, \tag{A.8}$$

$$c_x = r \cos \theta_C - a, \tag{A.9}$$

$$\theta_E = \text{polar\_angle}(c_y, c_x) \pm \arccos \frac{h}{\sqrt{c_x^2 + c_y^2}} - \frac{\pi}{2}, \tag{A.10}$$

$$\theta_C = \theta_E - \begin{cases} \arcsin\left(\dfrac{a}{r} \sin \theta_E - \dfrac{h}{r}\right), \\ \pi - \arcsin\left(\dfrac{a}{r} \sin \theta_E - \dfrac{h}{r},\right) \end{cases} \tag{A.11}$$

$$\frac{d\theta_C}{d\theta_E} = 1 \pm \frac{-a \cos \theta_E}{\sqrt{r^2 - (a \sin \theta_E - h)^2}}, \tag{A.12}$$

$$\frac{d^2\theta_C}{d\theta_E^2} = \pm \frac{a((r^2 - h^2 - a^2) \sin \theta_E + ah(1 + \sin^2 \theta_E))}{(r^2 - (a \sin \theta_E - h)^2)^{3/2}}, \tag{A.13}$$

Algorithm 18. *Find new contacts.*
Until no more interpolation is needed
  For each active INTERACTOR data structure (Fig. 9)
    For each pair of interacting primitive solids
      If Algorithm 9 finds an excessive depth of overlap
        Interpolate the state of the system to make the depth acceptable
        Restart this algorithm from the beginning

$$\frac{d\theta_E}{d\theta_C} = 1 + \frac{a}{c_x^2 + c_y^2}\left(c_x \pm \frac{hc_y}{\sqrt{c_x^2 + c_y^2 - h^2}}\right), \tag{A.14}$$

$$\frac{d^2\theta_E}{d\theta_C{}^2} = \frac{-a}{c_x^2 + c_y^2}\left(c_y + \frac{2ac_xc_y}{c_x^2 + c_y^2} \pm \frac{h}{\sqrt{c_x^2 + c_y^2 - h^2}}\left(\frac{2ac_y^2}{c_x^2 + c_y^2} + \frac{ac_y^2}{c_x^2 + c_y^2 - h^2} - c_x - a\right)\right). \tag{A.15}$$

A.2. Implementation

After each simulation step, (the model of) the machine being simulated will be in a new, previously unencountered, state. Therefore moving parts which were separated before may now be in contact. Actually, the parts will not, in general, be in perfect contact. Instead their volumes will overlap in space by some amount. The algorithm for finding new contacts must determine the depth of overlap and then interpolate the machine's state back to the point where the parts first came in contact. See Algorithm 18. Typically this algorithm will find only one new contact: the first to occur during the simulation step. The interpolation process will "re-separate" any other new contacts that have occurred.

A.2.1. Amalgamating kinematic subsystems

Elementary kinematic subsystems are represented by the KINEMATIC SUBSYSTEM data structure (Fig. 7). Kinematic subsystems are represented as directed trees of elementary kinematic subsystems. In my current implementation, an elementary kinematic subsystem may have one of three possible relationships with its parent: either a gear relation (see Section 2.1) or one of two types of transient higher pair relations. One type of transient higher pair relation is that of a corner of a block sliding along the edge of another block, as described in Section A.1.3. The other type of transient higher pair in the current implementation is a cylinder sliding along the edge of a block. This higher pair is equivalent to a different higher pair formed by extending the edge of the block outward by the radius of the cylinder and replacing the cylinder by a corner of a block at its center. Thus the problem of analyzing the second type of higher pair reduces to the problem of analyzing the first type, which was discussed in Section A.1.3.

Algorithm 19. *Amalgamate kinematic subsystems.*

1. Use Algorithm 20 to compute relationship data that will not change while the elements of the new transient higher pair remain in contact
2. Initialize the normal vector and contact point to the values returned by Algorithm 18
3. Initialize the first and second derivatives by plugging values directly into the derivative formulas in Section A.1.3
4. Compute the velocity of the new amalgamated kinematic subsystem using Eq. (A.19)

The KINEMATIC SUBSYSTEM data structure specifies the type of relationship with the parent and the associated relationship data. For a gear relation, the relationship data is just the gear ratio. For the transient higher pairs, some of the relationship data is constant as long as the pair exists:

(1) whether the child is a rotating line, or a rotating corner or cylinder,
(2) the cylinder radius (= 0 for a corner),
(3) the parameters a, r, and h (see Section A.1.3),
(4) C offset (θ_C = position state variable − C offset),
(5) E offset (θ_E = position state variable − E offset),
(6) coordinates of adjacent corners,
(7) coordinates of edge endpoints,

and some data changes with every simulation step:

(1) the normal vector at the point of contact,
(2) the point of contact (c_x, c_y) (see Section A.1.3),
(3) d(child position)/d(parent position),
(4) d^2(child position)/d(parent position)^2.

When two separate kinematic subsystems come in contact, my program uses Algorithm 19 to amalgamate them into a single subsystem. The unchanging data is computed using Algorithm 20. Kinematic pairs C and E (see Section A.1.3), which previously were elements of disjoint kinematic subsystems, must be linked into a single new subsystem. Since a kinematic subsystem is represented as a directed tree, if both kinematic pairs happen to be children in the trees representing their previous kinematic subsystems, then one of the trees must be reversed using Algorithm 21 so that the child becomes the root of the tree. On the other hand, if either pair is NOT already a child, then there is no reason to reverse a tree. However, if the fixed frame is part of a kinematic subsystem, it must always be the root of the tree, since its position cannot change and thus determines the positions of all the other parts.

A new contact begins with a collision. In Section A.1.2, I derived Eq. (A.7), which I restate here

$$(\dot{q}_j^{\text{after}} - \dot{q}_j^{\text{before}}) \sum_i I_i \left(\frac{dx_i}{dq_j}\right)^2 = \hat{F}\frac{dx_F}{dq_j} \tag{A.16}$$

I define q_j, the position state variable for kinematic subsystem j, to be the position state

Algorithm 20. *Find transient higher pair constant data.*

1) If either part is fixed
   then the other is the child—reverse its tree if necessary
   Else if the rotating corner (or cylinder) is not already a child
   then it is the child
   Else if the rotating edge is not already a child then it is the child
   Else reverse the rotating corner's tree, making it the root,
   and make it the child of the rotating edge
2) If the contact finder (Algorithm 18) found a cylinder
   then store its radius
3) Compute a, r, and h from the current positions of the parts (see Fig. A.2)
4) θ_C = polar_angle(contact point found by Algorithm 9)
   C offset = position state variable of C - θ_C
5) θ_E = polar_angle(normal vector found by Algorithm 9) $- \pi/2$
   E offset = position state variable of E - θ_E
6) transform corner and edge coordinates to the coordinate system
   of Fig. A.2

Algorithm 21. *Make elementary kinematic subsystem* EKS *the root of its tree.*

1. If *EKS* has a parent then apply this algorithm to it.
2. Copy all relationship data from *EKS* to its parent, which will now become its child, with the following changes:
   (a) negate the type flag (i.e. line versus corner),
   (b) $dx_{parent}/dx_{child} = 1/(dx_{child}/dx_{parent})$,
   (c) $d^2x_{parent}/dx^2_{child} = -d^2x_{child}/dx^2_{parent}/(dx_{child}/dx_{parent})^3$.

variable of the elementary kinematic subsystem which is the root of the tree representing kinematic subsystem j. For the geometry we are considering,

$$\frac{dx_F}{dq_j} = \boldsymbol{r}_j \times \boldsymbol{n}_j \cdot \hat{\boldsymbol{z}} \tag{A.17}$$

where $\boldsymbol{r}_j$ is the radius vector from the axis of the kinematic pair in kinematic subsystem j which has formed the new contact to the point of contact found by Algorithm 9, and $\boldsymbol{n}$ is the local surface normal found by Algorithm 9, which will be the direction of the impulse $\hat{\boldsymbol{F}}$. The sum on the left-hand side of Eq. (A.16) can be computed directly from the data in the tree of KINEMATIC SUBSYSTEM data structures. We may write an instance of Eq. (A.16) for each of the two colliding subsystems, and by Newton's third law of motion, if the collision force on one subsystem is $\boldsymbol{F}$, then the collision force on the other subsystem is $-\boldsymbol{F}$.

A third equation comes from the assumption that the collision is inelastic so that after the collision the two subsystems will be united into one. If the colliding subsystems are

Algorithm 22. *Reformulate the vector of state variables.*
state_variable_count := 0
For each elementary kinematic subsystem *EKS*
  If *EKS* does not have a parent
    then state_variable_vector[state_variable_count] := state(*EKS*)
      increment state_variable_count

j and k, then

$$\frac{\dot{q}_j^{\text{after}}}{\dot{q}_k^{\text{after}}} = \frac{\mathrm{d}q_j}{\mathrm{d}q_k}, \tag{A.18}$$

where $\mathrm{d}q_j/\mathrm{d}q_k$ is computable from the geometry of the newly amalgamated kinematic subsystem. These three equations can be solved algebraically to compute the three unknown values $\hat{F}$, $\dot{q}_j^{\text{after}}$, and $\dot{q}_k^{\text{after}}$. If q_j is the parent in the new amalgamated kinematic subsystem, then

$$\dot{q}_j^{\text{after}} = \frac{\dot{q}_j^{\text{before}} + \dfrac{\alpha_j}{\alpha_k}\dot{q}_k}{1 - \dfrac{\alpha_j}{\alpha_k}\dfrac{\mathrm{d}q_k}{\mathrm{d}q_j}}, \tag{A.19}$$

where

$$\alpha_m = \frac{\boldsymbol{r}_m \times \boldsymbol{n} \cdot \hat{\boldsymbol{z}}}{\sum_i I_i \left(\dfrac{\mathrm{d}x_i}{\mathrm{d}q_m}\right)^2}. \tag{A.20}$$

A.2.2. Simulation

Numerical simulation for this model is done by the publicly-available subroutine RKF45 [38], which my program calls once for each simulation time step. RKF45 is passed a vector of state variables specifying the current state of the machine and a pointer to one of my subroutines which it calls to compute the time derivatives of the state variables. RKF45 then returns a new vector of state variables specifying the new state of the system after the time step.

If my program finds new contacts using Algorithm 18, then the current set of kinematic subsystems will be changed using the algorithms in Section A.2.1. In this case, before calling RKF45 it is necessary to reformulate the vector of state variables using Algorithm 22, since the number of state variables will have changed. As I mentioned in Section A.2.1, I define q_j, the position state variable for kinematic subsystem j, to be the position state variable of the elementary kinematic subsystem which is the root of the tree representing kinematic subsystem j.

RKF45 calls a subroutine in my program which uses Algorithm 23 to compute the time derivatives of the state variables. The time derivatives of the position state variables are just the velocity state variables. The time derivatives of the velocity state variables

Algorithm 23. *Compute time derivatives of state variables.*
For each elementary kinematic subsystem *EKS*
If *EKS* does not have a parent
then position time derivative := velocity
numerator := 0
denominator := 0
Apply Algorithm 24 to *EKS*
velocity time derivative := *numerator* / *denominator*

Algorithm 24. *Recursive computations for elementary kinematic subsystem i.*
Let q_j be the position state variable of the kinematic subsystem containing i
If i does not have a parent
then $dx_i/dq_j = d^2x_i/dq_j^2 = 1$
else

1. Let elementary kinematic subsystem k be the parent of i
2. Compute x_i from x_k using Algorithm 25
3. Compute dx_i/dq_j by first using either Eq. (A.12) or Eq. (A.14) to compute dx_i/dx_k and then setting

$$\frac{dx_i}{dq_j} = \frac{dx_i}{dx_k}\frac{dx_k}{dq_j} \tag{A.21}$$

4. Compute d^2x_i/dq_j^2 by first using either Eq. (A.13) or Eq. (A.15) to compute d^2x_i/dx_k^2 and then setting

$$\frac{d^2x_i}{dq_j^2} = \frac{d^2x_i}{dx_k^2}\left(\frac{dx_k}{dq_j}\right)^2 \tag{A.22}$$

Add $-\dfrac{dx_i}{dq_j}\left(I_i\dfrac{d^2x_i}{dq_j^2}\dot{q}_j^2 + k_i x_i + h_i\dfrac{dx_i}{dq_j}\dot{q}_j\right)$ to *numerator*

Add $I_i(dx_i/dq_j)^2$ to *denominator*
For each *CHILD* of elementary kinematic subsystem i
Apply this algorithm to *CHILD*

Algorithm 25. *Compute position of child from parent.*
If the child is a moving corner (or cylinder)
then compute x_i from x_k using Eq. (A.11)
else compute x_i and (c_x, c_y) from x_k using Eqs. (A.8), (A.9), and (A.10)
Choose the value of x_i closest to the previous value

Algorithm 26. *Fill in the new nonconstant values.*
For each elementary kinematic subsystem *EKS*
If *EKS* does not have a parent
then for each *CHILD* of *EKS*
Apply Algorithm 27 to *CHILD*

Algorithm 27. *Recursively fill in the new nonconstant values.*
1. Let elementary kinematic subsystem k be the parent of i
2. Compute x_i and (c_x, c_y) from x_k using Algorithm 25
3. normal vector := $(-\sin(\theta_E), \cos(\theta_E))$
4. Compute dx_i/dx_k from x_k using either Eq. (A.12) or Eq. (A.14)
5. Compute d^2x_i/dx_k^2 using either Eq. (A.13) or Eq. (A.15)

are computed from Eq. (A.5) by Algorithm 23 (which calls Algorithm 24). Note that all positions and first and second derivatives are recomputed from scratch. This is necessary since when RKF45 calls my time-derivative subroutine it passes a state variable vector for some potential future state of the mechanism, not one that has been seen before. Therefore all other quantities must be recomputed from this new vector.

Algorithm 24 uses Algorithm 25 to compute the position of the child subsystem from that of the parent. Since the appropriate equations from Section A.1.3 always give two possible values, Algorithm 25 must choose between them. In the current implementation, this is done by simply choosing whichever of the two values is closer to the value used on the previous step. Since the two values are typically quite far apart, this heuristic works well in practice. The first time this choice is made for a particular transient higher pair, the previous value used as a guide is the one returned by the contact finder and thus is not ambiguous. The correctness of the choice is checked by two other methods; warning messages are printed in case of disagreement. The first of the other methods is to check whether one of the possible choices causes the corner to be past the end of the edge. The second method is to check whether the position change is consistent with the velocity. Often these two methods aren't able to make a choice, which is why neither is the primary method.

RKF45 returns a vector of state variables representing the new state of the machine after the time step. My program then uses Algorithm 26 to fill in the new nonconstant values for every elementary kinematic subsystem.

A.2.3. Detecting separation

After the state of the machine has advanced by one simulation time step, as described in Section A.2.2, the current set of valid kinematic subsystems may have changed. In Section A.2.1, I discussed the amalgamation of kinematic subsystems after a time step as a result of new contacts detected by Algorithm 18. Configuration changes after a simulation time step may also cause parts to separate which were previously in contact.

Algorithm 28. *Separate kinematic subsystems.*
For each elementary kinematic subsystem *EKS*
If *EKS* has a parent
AND the point of contact between *EKS* and its parent is not between the endpoints of the edge
OR one of the two edges ending at the corner is collinear with the edge that the corner is sliding on
then break the contact between *EKS* and its parent

In that case, the tree of KINEMATIC SUBSYSTEM data structures representing the kinematic subsystem must be broken into two separate trees by removing the transient higher pair which is no longer valid.

Separation of invalid transient higher pairs is done by applying Algorithm 28 to every elementary kinematic subsystem. There are two ways in which a transient higher pair may become invalid. If a corner or a cylinder is sliding along an edge, it may fall off the end of the edge. Alternatively, if a corner is sliding along an edge, relative rotation of the two parts may convert the corner-to-edge contact into an edge-to-edge contact. Typically, this contact will then immediately turn into another corner-to-edge contact involving a different corner and/or edge. Therefore, although the two parts will still be linked by a higher pair, the old higher pair is no longer valid. In this case, Algorithm 28 deletes the old higher pair, and if a new higher pair needs to be instantiated, that is done in the usual way using Algorithm 19 after the new contact has been detected using Algorithm 18.

References

[1] A.A. Andronov, A.A. Vitt and S.E. Khaikin, *Theory of Oscillators* (Pergamon Press, Oxford, England, 1966).
[2] F.P. Bowden and D. Tabor, *The Friction and Lubrication of Solids* (Oxford University Press, London, 1950).
[3] C. Conti, P. Corron and P. Michotte, A computer-aided kinematic analysis system for mechanism design and computer simulation, *Mechanism Mach. Theory* **27** (5) (1992) 563–574.
[4] J.F. Cremer, An architecture for general purpose physical system simulation—integrating geometry, dynamics, and control, Ph.D. Thesis, Department of Computer Science, Cornell University, Ithaca, NY (1989).
[5] P.A. Cundall, Formulation of a three-dimensional distinct element model—part I: a scheme to detect and represent contacts in a system composed of many polyhedral blocks, *Int. J. Rock Mech. Mining Sci. Geomech. Abstracts* **25** (3) (1988) 107–116.
[6] G. Dawson, The dynamic duo: Dram and Adams, *Comput. Mech. Eng.* (1985).
[7] P.G. Drazin, *Nonlinear Systems* (Cambridge University Press, Cambridge, England, 1992).
[8] B. Fallahi and K.M. Ragsdell, A compact approach to planar kinematic analysis, *Trans. ASME J. Mechanisms Transmissions Automation in Design* **105** (1983) 434–440.
[9] K. Forbus, Qualitative process theory, *Artif. Intell.* **24** (1984) 85–168.
[10] K.D. Forbus and B. Falkenhainer, Self-explanatory simulations: An integration of qualitative and quantitative knowledge, in: *Proceedings AAAI-90*, Boston, MA (1990).

[11] K.D. Forbus, P. Nielson and B. Faltings, Qualitative spatial reasoning: the CLOCK project, *Artif. Intell.* **51** (1991) 417–471.
[12] C.W. Gear, *Numerical Initial Value Problems in Ordinary Differential Equations* (Prentice-Hall, Englewood Cliffs, NJ, 1971).
[13] A. Gelsey, Automated reasoning about machine geometry and kinematics, in: *Proceedings Third IEEE Conference on Artificial Intelligence Applications*, Orlando, FL (1987); also in: D.S. Weld and J. de Kleer, eds., *Readings in Qualitative Reasoning about Physical Systems* (Morgan Kaufmann, San Mateo, CA, 1990).
[14] A. Gelsey, Automated physical modeling, in: *Proceedings IJCAI-89*, Detroit, MI (1989).
[15] A. Gelsey, Automated reasoning about machines, Ph.D. Thesis, YALEU/CSD/RR#785, Yale University, New Haven, CT (1990).
[16] A. Gelsey, Using intelligently controlled simulation to predict a machine's long-term behavior, in: *Proceedings AAAI-91*, Cambridge, MA (1991) 880–887.
[17] B.J. Gilmore and R.J. Cipra, Simulation of planar dynamic mechanical systems with changing topologies, *J. Mech. Design* **113** (1991) 70–83.
[18] H. Goldstein, *Classical Mechanics* (Addison-Wesley, Reading, MA, 2nd ed., 1980).
[19] S. Goyal, F.W. Sinden and E.N. Pinson, Simulation of dynamics of rigid bodies I, II, and III, in: *Proceedings Pacific-Rim International Conference on Modelling, Simulation and Identification*, Vancouver, BC (1992).
[20] G. Hartquist, Public PADL-2, *IEEE Comput. Graph. Appl.* (1983) 30–31.
[21] E.J. Haug, A survey of dynamics software, in: E.J. Haug, ed., *Computer Aided Analysis and Optimization of Mechanical System Dynamics* (Springer-Verlag, Berlin, 1984) 24–31.
[22] E.J. Haug, *Computer Aided Kinematics and Dynamics of Mechanical Systems, Volume* 1: *Basic Methods* (Allyn and Bacon, Boston, MA, 1989).
[23] A.C. Hindmarsh, ODEPACK, a systematized collection of ODE solvers, in: R. Stepleman, editor, *Scientific Computing: Applications of Mathematics and Computing to the Physical Sciences*, IMACS Transactions on Scientific Computation (North-Holland, Amsterdam, 1983) 55–64.
[24] C.M. Hoffmann, *Geometric and Solid Modeling: An Introduction* (Morgan Kaufmann, San Mateo, CA, 1989).
[25] L. Joskowicz and E.P. Sacks, Computational kinematics, *Artif. Intell.* **51** (1991) 381–416.
[26] A.J. Lichtenberg and M.A. Lieberman, *Regular and Stochastic Motion* (Springer-Verlag, Berlin, 1983).
[27] N. Minorsky, *Nonlinear Oscillations* (D. Van Nostrand, Princeton, NJ, 1962).
[28] A. Nakamura and N. Nakajima, Computer-aided design diagnosis for machines—kinematic model extraction from mechanisms, *Mech. Mach. Theory* **25** (3) (1990) 355–364.
[29] N. Orlandea, M.A. Chace and D.A. Calahan, A sparsity-oriented approach to the dynamic analysis and design of mechanical systems, *J. Eng. Industry* **99** (1977) 773–784,
[30] B. Paul, *Kinematics and Dynamics of Planar Machinery* (Prentice-Hall, Englewood Cliffs, NJ, 1979).
[31] A.B. Pippard, *The Physics of Vibration* (Cambridge University Press, Cambridge, England, 1978).
[32] C.D. Potter, Mechanism analysis moves with the times, *Comput. Graph. World* **15** (5) (1992) 30.
[33] A.A.G. Requicha, Representations for rigid solids: theory, methods, and systems, *ACM Comput. Surv.* **12** (1980) 437–464.
[34] F. Reuleaux, *The Kinematics of Machinery* (Macmillan, London, 1876).
[35] E.J. Routh, *The Elementary Part of a Treatise on the Dynamics of a System of Rigid Bodies* (Dover, New York, 1960).
[36] E.P. Sacks, Automatic analysis of one-parameter ordinary differential equations by intelligent numeric simulation, *Artif. Intell.* **48**(1) (1991).
[37] E.P. Sacks and L. Joskowicz, Automated modeling and kinematic simulation of mechanisms, *Comput.-Aided Design* **25** (2) (1993) 106–118.
[38] L.F. Shampine, H.A. Watts and S. Davenport, Solving non-stiff ordinary differential equations—the state of the art, *SIAM Rev.* **18** (1976) 376–411.
[39] D.S. Weld, The use of aggregation in causal simulation, *Artif. Intell.* **30** (1986) 1–34.
[40] J. Wittenburg, *Dynamics of Systems of Rigid Bodies* (Teubner, Stuttgart, Germany, 1977).
[41] K.M.K. Yip, Understanding complex dynamics by visual and symbolic reasoning, *Artif. Intell.* **51** (1–3) (1991) 179–221.