Compositional Modeling for Complex Spatial Reasoning Tasks*

Kyungsook Han and **Andrew Gelsey**
kshan@cs.rutgers.edu gelsey@cs.rutgers.edu
Department of Computer Science
Rutgers University
New Brunswick, NJ 08903
U. S. A.

Abstract

Reasoning about a complex physical system generally requires the creation and execution of a model of the system, the creation of which in turn depends on the types of knowledge available for the physical system and their representation. Such a model is normally created by the person studying the system. Despite the considerable time and effort spent, a hand-crafted model is often error-prone. Modifying a hand-crafted model to solve a similar problem about other physical systems is also difficult, and may take more time than building a new model for the systems. We describe a method which uses first principles to automatically create models and simulators for spatially complex motions. This method solves several problems with existing AI modeling work on motion by: (1) explicit handling of vector quantities and frames of reference; (2) simultaneous handling of multiple equations (algebraic or differential, linear or nonlinear); and (3) declarative, algorithm-neutral representation of physics knowledge. The method has been implemented in a working program called ORACLE and tested in the domains of mechanical devices and sailboats. Experimental results show that ORACLE is capable of generating correct models of several different types of physical systems if enough domain knowledge is available.

Introduction

Spatial reasoning problems are considered in a variety of areas, but different areas have different spatial reasoning tasks. In computer vision, for example, recognition of familiar objects can be the main spatial reasoning task. Our focus in this paper is on modeling physical systems for reasoning about spatially complex motion of the systems. By spatially complex motion we mean the motion of multiple moving objects in arbitrary configurations in three dimension. Motion in three dimension is much more difficult to understand and deal with than the motion in one or two dimension because modeling such a motion involves reasoning in vector space.

There are several works on modeling motion, but spatially complex motions did not get much attention. Some qualitative physics approaches have been used to model motion. But they focus on developing representations for physical systems and reasoning about the systems within the representations, and their capability of reasoning about motion is limited to simple motions only. Consider, for example, a spring with one end attached to a fixed point and the other end attached to a block. What happens if the block is pulled *and* rotated from its equilibrium position and released, as illustrated in Figure 1a? This spring-block system is different from a linear harmonic oscillator, which is a common textbook example often used in qualitative physics research. It is well known that the harmonic oscillator has one degree of freedom, i.e., displacement of the block from its equilibrium position, and its motion is oscillatory on a straight line. However, predicting the behavior of the spring-block system in Figure 1a is not as simple as the harmonic oscillator. Many qualitative physics approaches which can model the harmonic oscillator (Forbus 1984; Kuipers 1986; Struss 1988; Weld 1988; Williams 1986) cannot handle this spring-block system. The primary reason for this difficulty can be attributed to the fact that qualitative values are not adequate for spatially complex problems and that they lack the ability to reason explicitly about vector quantities and moving frames of reference. Some research in spatial reasoning or commercial mechanics simulators have powerful algorithms to model motion, but they incorporate knowledge of physical phenomena directly into algorithms and difficult to reuse or extend them to solve similar problems about other physical systems.

The work of this paper is motivated by two goals. The first goal is to automate the model formulation and simulation process for complex motion. The second goal is to make the modeling process as general as possible so that common domain theories can be shared and reused instead of being duplicated. As we

*This research was partially supported by the Advanced Research Projects Agency (ARPA) and National Aeronautics and Space Administration under NASA grant NAG2-645 and by the National Science Foundation through grant CCR-9209793.

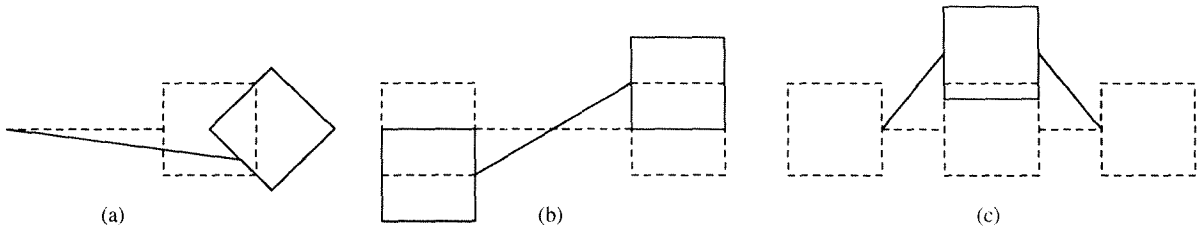

Figure 1: (a)1 block attached to a spring. The block is pulled and rotated from its equilibrium position and released. (b) 2 blocks connected by 1 spring. Both blocks are pulled in opposite directions and released. (c) 3 blocks connected by 2 springs. The middle block is pulled directly to the side and released.

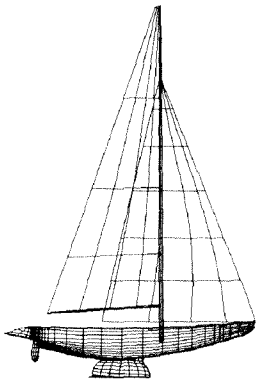

Figure 2: *Stars & Stripes*, winner of the 1987 America's Cup competition.

will show later, we handle explicitly vector quantities and reference frames, and construct a model from a set of model fragments. The model fragments represent the fundamental physics knowledge in a declarative and algorithm-independent way, and are reused in building models of different types of physical systems.

The remainder of this paper discusses a framework for automatically creating and simulating behavioral models of moving physical systems, and illustrates the framework using the single spring-block system in Figure 1a as a running example. We also discuss the epistemological adequacy of the framework for broader class of physical systems, and address related issues such as: Can the modeling system of the single spring-block system be used to predict the behavior of more general spring-block systems such as Figure 1b and Figure 1c, or different types of physical systems such as a sailboat shown in Figure 2? Do we need a separate modeling system for every physical system? Or the same modeling system just with more knowledge will suffice to model them?

Framework for Model Building and Simulation

In this section we describe the framework of our modeling system called ORACLE, implemented in the mathematical manipulation language Maple (Char *et al.* 1991).

Ontology and Representation

The principal elements of ORACLE's ontology are entities, phenomena, model fragments, and models, each represented in a frame (Minsky 1975). An *entity* is a physical object which either constitutes a physical system by itself (i.e., primitive object) or is a part of a physical system (i.e., composite object). The properties of an entity are expressed as variables in equations. The block entity, for example, has properties such as position and velocity. An entity is represented in a frame with slots for the properties. Facets allowed in a slot are value, form, range, if_needed, and if_added. The value facet is initially set to null but will be assigned a vector, scalar, string, set, or any other expression as it becomes known. The form facet distinguishes the slot type (e.g., scalar or vector) and is consulted when the system creates a new Maple variable name during the problem solving process. For example, if the system is asked to compute the position of a block b1, a set of new variables {b1x(t), b1y(t), b1z(t)} will be created for the position vector and used in equations. The range facet specifies a valid range of the property value if it is known. The if_needed facet or if_added facet holds the procedure call, invoked when a slot value is needed or added. The if_needed procedure of the velocity slot in the example below says that velocity is derivable from position.

```
block=[AKO=rigid_body,
  position(t)=[value=null, form=[x(t),y(t),z(t)]],
  velocity(t)=[value=null, form=[u(t),v(t),w(t)],
    if_needed=[derive_velocity, position(t)]],
        ....  (other slots not shown)  .... ]
```

A *phenomenon* is a process which changes one or more properties of an entity in a physical system. Force from a spring, for example, is a phenomenon which can change the position and/or orientation of an entity which is attached to the spring.

A *model fragment* is a characterization of a physical phenomenon by a set of entities, variables, assumptions, and equations. There may be more than one model fragments for a single phenomenon, each with different assumption or approximation. The equations of a model fragment are applicable when the corre-

sponding phenomenon occurs. The spring force, for example, exerted on an object attached to end2 of a linear spring with linear damping is represented as follows (syntax slightly modified for readability):

```
Springforce2=[phenomenon='spring force at end2',
  entities=[s=linear_damped_spring],
  variables=[k=s[force_const],
    b=s[damping_coeff],
    e1(t)=s[end1(t)], e2(t)=s[end2(t)],
    l=s[rest_length], f(t)=s[force2(t)]],
  equations=[f(t)=-k*(||e2(t)-e1(t)||-l)*
    (e2(t)-e1(t))/||e2(t)-e1(t)||-
    b*diff((||e2(t)-e1(t)||-l)*(e2(t)-e1(t))/
    ||e2(t)-e1(t)||,t)]]
```

It says that s is a linear damped spring, k is a force constant of the spring, b is a damping coefficient, e1(t) and e2(t) are the position vectors of end1 and end2, l is the rest length, and f(t) is the spring force at end2. $\| e2(t) - e1(t) \|$ is the vector norm representing the length of the spring at time t, $\| e2(t) - e1(t) \| - l$ is the signed length change from the rest length, and $(e2(t) - e1(t))/ \| e2(t) - e1(t) \|$ is a unit vector with direction from end1 to end2.

A *model* is a composition of model fragments applicable to a physical system in a particular situation. *Simulation* is the execution of a model.

The motion of an entity at any instant can be described by a set of ordinary differential equations in the twelve components of four vectors: position, orientation, velocity, and angular velocity.[1] The differential equations are usually nonlinear and do not have a solution in closed form, so they must be solved by numeric integration. For a moving entity, ORACLE constructs a model with the twelve components of the four vectors (position, orientation, velocity, and angular velocity) as *state variables*, which take numeric values during simulation.

The state variables of each subpart of an entity are initially defined in the local reference frame, which is assumed to be fixed to the entity. Then each subpart defined in its local reference frame is translated and rotated by having its reference frame redefined in a common inertial reference frame. The system chooses the common inertial reference frame from local reference frames which are not accelerated. If there is no such reference frame (i.e., all the local reference frames are noninertial), it introduces a new inertial reference frame. If there are several inertial reference frames, the choice is arbitrary.

[1] The degrees of freedom of a moving entity are six instead of twelve because the velocity function and the angular velocity function are derivable by differentiating the position and orientation functions, respectively. The motion of a physical system with n subparts can be characterized by maximum $12n$ state variables with $6n$ degrees of freedom.

Algorithm

ORACLE takes as input a description of a physical system in terms of the entities of the system, any constraints to be satisfied, and the properties of the entities (i.e., variables) whose values are going to be computed by modeling and simulation. As output, it produces a model of the motion of the system and the variable values obtained by solving the model. The algorithm of ORACLE consists of three phases: (1) problem analysis, (2) model creation, and (3) model execution. In the first phase, ORACLE represents each entity of a problem statement in a frame by copying a class frame and filling in slots for property values specified in the

---

Algorithm 1 ORACLE's top-level algorithm

Problem Analysis Analyze a problem statement.

1. Analyze entities, and create frames of the entities and a set INIT of initial-value conditions.
2. For each constraint, determine its type and represent them in equations.
3. Analyze variables and generate a set DRVD of differential equations.

Model Creation Search for relevant model fragments and compose a behavioral model with them.

1. For each entity E of the problem statement
   For each model fragment MF indexed by the "mf" slot of E
   If MF has not been instantiated for E AND every variable of MF either corresponds to an entity property or variable of the input or can be derived from them AND the assumption (if any) of MF does not violate any entity property or constraint of the input
   Put MF in a list MFS.
2. model M = DRVD
3. #equations = #equations(M)
4. retry: For each model fragment MF in MFS
   (a) Instantiate MF for the problem.
   (b) $M = M \cup \{MF\}$
   (c) #equations = #equations(M)
   (d) If #equations = #variables, do **model execution.**
5. Print the dead-end situation, and quit.

Model Execution Solve the model M either analytically or by numeric simulation.

1. Determine the types of equations of the model and solve them with INIT for the variables.
2. If a valid solution is obtained, print the model and solutions, and quit.
3. If a valid solution is not obtained, retract the most recent MF from the model and go to retry.

---

problem statement. It also transforms vector quantities expressed in the local reference frames into those in the inertial reference frame, formulates initial conditions, and executes if_added procedures in the slots. A model fragment specifying forces on a component of a composite object is instantiated by if_added procedures in the this phase. After constraints are analyzed, variables are examined to determine if their values are already known in their slot values or derivable from other variables. In the second phase, additional model fragments which have not been instantiated are retrieved and a model is constructed from them. In the final phase, the constructed model is solved for the problem. If ORACLE runs out of potentially relevant model fragments before it finds a valid solution, it prints the situation, asks more information, and quits. The top-level algorithm of ORACLE is outlined in Algorithm 1.

Example

We illustrate how ORACLE works with the spring-block system of Figure 1a. Suppose the following problem description is given as an input. There is no particular constraint in this problem and the system is asked to compute the four vector variables (twelve variables in component form) of the block.

```
entities=[b1=[block, mass=1,
    principal_moments_of_inertia=[1/6,1/6,1/6],
    position(0)=[3,0,0],
    orientation(0)=[Pi/4,Pi/2,0],
    velocity(0)=[0,0,0],
    ang_velocity(0)=[0,0,0]],
  s1=[spring, force_const=10,
      damping_coeff=1/10, rest_length=3/2,
      end1(t)=[0,0,0], end2(t)=b1[-1/2,0,0]],
  sb=[composite_object, parts={b1,s1}]];
constraints=[ ];
variables=[b1[position(t)], b1[orientation(t)],
           b1[velocity(t)], b1[ang_velocity(t)]];
```

For each entity b1, s1, and sb, a frame is created and the given properties of the entities are recorded in their slot values. The if_added procedure in the end2 slot of s1 computes the spring force acting on b1 using a model fragment Springforce2 and records the value in the force slot of b1. The position of end2 in the inertial reference frame is computed from a translation and a rotation of the local reference frame of b1. The initial conditions of the block are also formulated. Here are the Maple variable names ORACLE assigns for this example.

```
INIT =
{b1x(0)=3, b1y(0)=0, b1z(0)=0,
 b1phi(0)=0, b1theta(0)=Pi/2, b1psi(0)=0,
 b1u(0)=0, b1v(0)=0, b1w(0)=0,
 b1omega1(0)=0, b1omega2(0)=0, b1omega3(0)=0}
```

None of the four state variables of b1 can be assigned a value simply by looking at slot values of b1, but the velocity and the angular velocity functions can be derived by differentiating the position and the orientation functions, respectively, according to their if_needed facets. The system generates trivial differential equations for the velocity and angular velocity by the procedures attached to the if_needed facets.

```
DRVD =
{b1u(t)=diff(b1x(t),t),
 b1v(t)=diff(b1y(t),t),
 b1w(t)=diff(b1z(t),t),
 b1omega1(t)=diff(b1theta(t),t)*cos(b1phi(t))+
   diff(b1psi(t),t)*sin(b1theta(t))*sin(b1phi(t)),
 b1omega2(t)=diff(b1theta(t),t)*sin(b1phi(t))-
   diff(b1psi(t),t)*sin(b1theta(t))*cos(b1phi(t)),
 b1omega3(t)=diff(b1phi(t),t)+
   diff(b1psi(t),t)*cos(b1theta(t))}
```

Now ORACLE focuses on finding equations for the position and the orientation. The equations for them cannot be derived from other variables since they are basic variables, so ORACLE looks for relevant model fragments. It examines model fragments, indexed by the mf slot of the block. ORACLE decides that Newton2 and Euler are potentially relevant because the entities (solid and rigid body, respectively) of the model fragments are superclass of a block and the equations of the model fragments contain at least one variable of the problem. Model fragments of Newton2 and Euler are as follows.

```
Newton2=[phenomenon='Newton's second law
                     of motion',
         entities=[r=solid],
         variables=[f(t)=r[net_force(t)],
                    p(t)=r[momentum(t)]],
         assumptions=[ ],
         equations=[f(t)=diff(p(t), t)]]

Euler=[phenomenon='time-dependency of
                   ang_velocity',
       entities=[b=rigid_body],
       variables=[Omega(t)=b[ang_velocity(t)],
                  M(t)=b[ang_momentum(t)],
                  T(t)=b[net_torque(t)]],
       assumptions=[ ],
       equations=[add(diff(M(t), t),
                  crossprod(Omega(t),
                  M(t)))=T(t)]
```

The entity and variable names of the model fragments are instantiated as those of the problem and they are substituted in the equations of the model fragments. The angular momentum is derived from principal moments of inertia and angular velocity by the if_needed procedure in the ang_momentum slot. Likewise, the net torque is derived from force and position vector of the point at which the force acts.

$$M(t) = \begin{pmatrix} I_1\Omega_1(t) \\ I_2\Omega_2(t) \\ I_3\Omega_3(t) \end{pmatrix}, \quad T(t) = \sum r \times f(t)$$

The principal moments of inertia (I_1, I_2, I_3) and the position vector (r) of the spring-attached point can be assigned from the information of the problem description. The angular velocity is one of the state variables

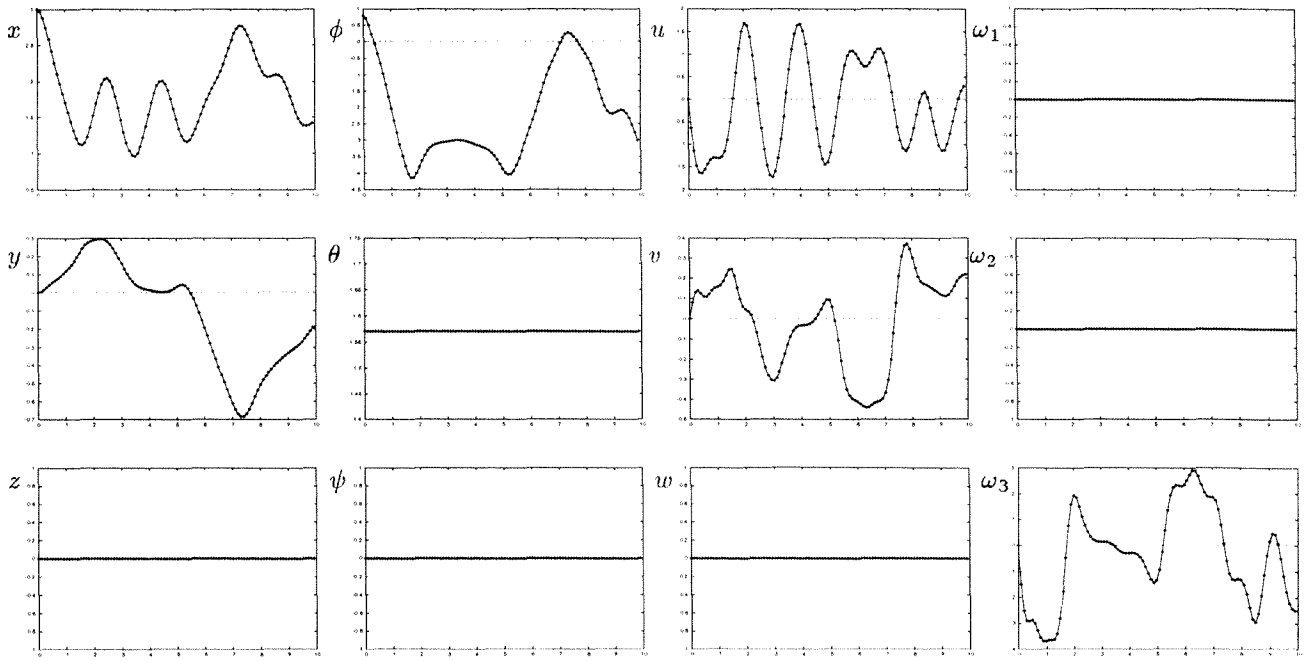

Figure 3: Plots of the 12 state variables of the block b1 as functions of time. The first column shows the position vector, the second column the orientation vector, the third column the velocity, and the last column the angular velocity.

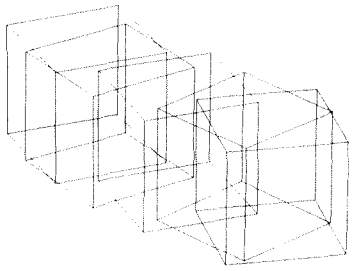

Figure 4: Motion of the block b1. Spring not shown.

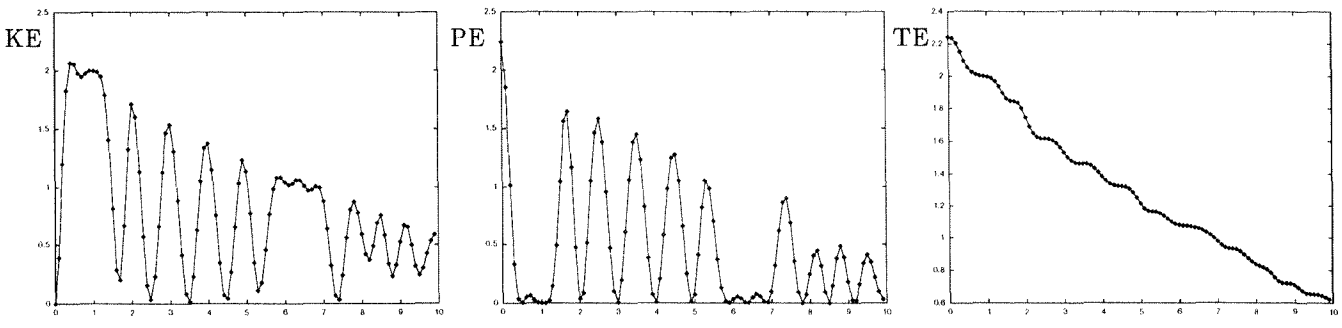

Figure 5: The kinetic, potential, and total energy of the single spring-block system as functions of time during the simulation.

asked by the problem, and its functions are derived in DRVD. The value of force $f(t)$, which has computed using a model fragment Springforce2 (shown earlier in section), is available in the force slot of b1, and substituted in the equations of Newton2 and Euler.

The system has now total 12 equations in component forms (6 from the model fragments and 6 from DRVD) plus 12 initial conditions for 12 unknowns. Several of the differential equations are nonlinear, and when ORACLE attempts to solve the model analytically, it does not find a solution in closed form. ORACLE then solves the differential equations by numeric simulation. ORACLE displays the simulation result by showing the state variables as functions of time using gnuplot (Figure 3). Animation of the moving block is shown (Figure 4) using PADL-2 solid modeling system (Hartquist 1983). Figure 4 contains several animation scenes superimposed. Note that the motion of the block is much more complex than that of the linear harmonic oscillator. The kinetic energy, potential energy, and total energy of the system are also displayed as part of validation criteria of the results (Figure 5). The total energy in Figure 5 decreases over time due to the nonzero damping coefficient of the spring of the problem statement.

Extension to Broader Class of Physical Systems

Multiple Spring-block Systems

The previous section showed how ORACLE predicts the behavior of the single spring-block system. Can the modeling system of the single spring-block system be used to predict the behavior of the multiple spring-block systems such as Figure 1b and Figure 1c? The answer is "yes". The multiple spring-block systems have additional entities and phenomena, but they are simply the multiple occurrences of the same types as the single spring-block system. Having already enough knowledge represented in general form to handle the single spring-block system, ORACLE can handle the multiple spring-block systems with no change. The way it solves the problem is the same. It computes the positions of ends of each spring in the inertial reference frame by transforming the local reference frame of its associated block, and derives differential equations for the velocity and angular velocity of the blocks from the procedures attached to the if_needed facets. It then instantiates model fragments of Spring force, Newton2 and Euler, and composes a model. Notice that model fragment sharing occurs within the models because each of those model fragments is instantiated more than once for different entities. The result of the execution of the models indicate that although none of the blocks are initially rotated, both blocks of Figure 1b and the end blocks of Figure 1c rotate as well as translate due to spring forces which are not parallel to the radius vectors of the points to which the springs are attached. If the spring damping is ignored (i.e., damping_coeff = 0), the middle block of Figure 1c shows translational motion only, but it shows both translational and rotational motions if the spring damping is considered (damping_coeff $\neq$ 0). Figure 6 shows part of animation scenes for the case when the spring damping is considered. In fact ORACLE is able to handle multiple rigid bodies connected by springs in arbitrary positions and orientations because the way of identifying relevant model fragments and composing them is not restricted by the number of entities or their connections.

Sailboat in Fluid

A sailboat is a composite object whose driving force comes from the differential motion of air over water. Before we model the sailboat, we can ask the same question as before. Can we use the modeling system of the spring-block systems to predict the behavior of a sailboat in fluids? The answer is "yes", provided that the modeling system has enough domain knowledge to handle the problem. We do not need to build a different modeling system. A modeling system with the same algorithm and the same model fragments plus additional model fragments and entities can predict the behavior of the sailboat.

New classes of entities added to the knowledge base are fluids (water and air) and lifting surfaces (hull and sail). The sailboat, water, and air entity have their own reference frames, which move as their entities move. New phenomena include hydrodynamic and aerodynamic forces, each with two components (lift and drag), and skin friction. A single model fragment is used to represent both hydrodynamic and aerodynamic frictional drag forces, and later instantiated for them. Likewise, a single model is used to represent both hydrodynamic and aerodynamic lift forces.

```
FDrag=[phenomenon='frictional drag force on
        an object in fluid',
    entities=[s=physical_object, f=fluid],
    variables=[FD=s[fdrag(t)],
        v=s[rel_fluid_speed(t)],
        fd=s[rel_fluid_direction(t)],
        Pa=s[parasitic_area],
        rho=f[density]],
    assumptions=[ ],
    equations=[FD=1/2*Pa*rho*v^2*fd]]

Lift=[phenomenon='lift and lift induced force on
        an object in fluid',
    entities=[s=physical_object, f=fluid],
    variables=[LF=s[lift(t)],
        L=s[lift_magnitude(t)],
        v=s[rel_fluid_speed(t)],
        fd=s[rel_fluid_direction(t)],
        pd=s[perpendicular_rel_fluid_dir(t)],
        Ca=s[effective_capture_area],
        rho=f[density]],
    assumptions=[s[rel_fluid_speed(t)] > 0],
    equations=[LF=L*pd+L^2/(2*Ca*rho*v^2)*fd]]
```

For the hull, the rel_fluid_speed is the speed of the boat relative to water. For the sail, the rel_fluid_speed

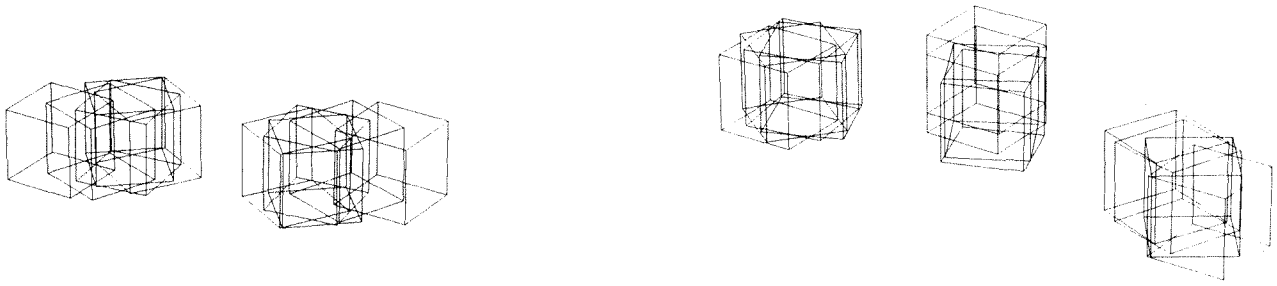

Figure 6: Motion of the multiple spring-block systems in Figure 1b and Figure 1c. Springs not shown.

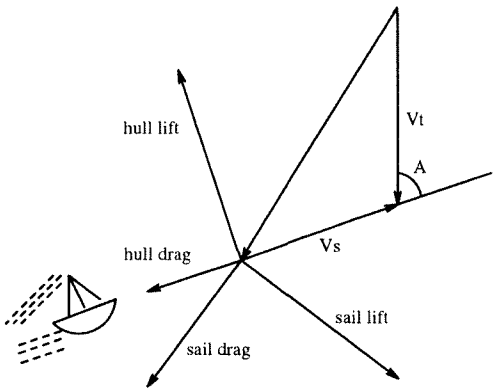

Figure 7: Directions of force components on a sailboat, adapted from (Letcher, 1976)

is the speed of the boat relative to air. The magnitude and direction of the hydrodynamic forces acting on the hull depend on the rel_fluid_speed of the hull, and the aerodynamic forces depend on the rel_fluid_speed of the sail. rel_fluid_direction is an angle between the direction of fluid and the direction of an object, represented in angle. Drag forces acting on the hull have components opposite to the direction of its rel_fluid_speed. Lift forces on the hull are perpendicular to the direction of its rel_fluid_speed. Similarly, drag forces on the sail have components opposite to the direction of its rel_fluid_speed, and lift forces on the sail are perpendicular to the direction of its rel_fluid_speed. Lift forces on the hull and sail of a sailboat are horizontal, not vertical.

The directions of the force components are summarized in Figure 7. In the figure, water is assumed to be at rest (i.e., speed = 0) and V_t, V_s, and A denote the wind speed, sailboat speed, and course angle from the wind direction, respectively.

Notice that the model fragment Lift has a nonempty assumption slot, saying that the relative fluid speed must be positive. When both the fluid and the object are at rest or the fluid has the same speed as the object in the same direction, the relative fluid speed becomes zero and the equations of the model fragment cannot be defined due to zero denominator. Thus we have another model fragment of lift with a different assumption slot; it says that lift force is zero when the relative fluid speed is zero.

```
Lift_at_zero_speed=[
     phenomenon='lift and lift induced
             force on an object in fluid',
     entities=[s=physical_object, f=fluid],
     variables=[L=s[lift(t)],
                sv=s[rel_fluid_speed(t)]],
     assumptions=[s[rel_fluid_speed(t)] = 0],
     equations=[L[1]=0,
                L[2]=0,
                L[3]=0]]
```

During problem solving, ORACLE automatically chooses between the two model fragments of lift by checking their assumptions. At present, the kinds of assumptions ORACLE can process are confined to the algebraic properties of entities, such as numeric ranges or arithmetic expressions.

After the entities and model fragments are added, ORACLE can solve several types of problems on a sailboat, but we will focus on one type of problem in this section. Suppose that a sailboat is heading in the angle of 49 degrees from the direction of wind at uniform speed 16.9 ft/sec and that water is at rest. The system is asked to compute the sailboat speed which will balance all the forces involved.

ORACLE first infers all the forces on the sailboat from the forces acting on its components, hull and sail. It instantiates the model fragments FDrag and Lift for each of them and records the summation of them in the net_force slot of the sailboat.

$$\mathrm{F} = \sum_{\mathrm{i} \in \{\mathrm{hull,\ sail}\}} (\mathrm{FDrag_i} + \mathrm{Lift_i})$$

It then searches for a model fragment which relates forces with speed, and finds the model fragment of Newton2. It substitutes the equations of the forces in the equation of Newton2, F(t) = d(p(t))/dt. Since the problem states that all the forces are balanced, the net force on the sailboat must be zero, implying the momentum p(t) is constant. The right hand side of the equation becomes zero from the constant momentum, resulting in an algebraic equation. However, the

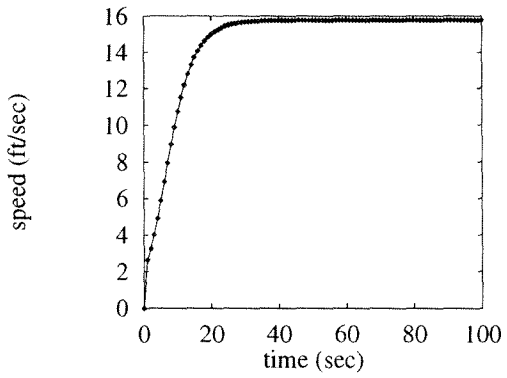

Figure 8: The sailboat speed as a function of time.

problem is under constrained in the sense that total number of equations in component form is 2 (Fx=0, Fy=0, Fz becomes a trivial equation 0=0) but the total number of unknowns in the equations is 3 (boat speed, sail lift magnitude, and hull lift magnitude). ORACLE prints the situation, asking for further information. The user provides an additional equation, $\partial(\mathrm{Fx})/\partial(\mathrm{sail\_lift\_magnitude}) = 0$, by making a simplifying assumption that the sail is controlled as to maximize the sailboat force in the direction of boat heading. The equations are solved algebraically, producing a solution, boat speed = 15.7 ft/sec. The solution is checked against the range facet of the speed slot of the boat, which says that value of the boat speed must be in the range of [0 .. infinity]. Since the solution of 15.8 is included in the range, it is returned to the user as a valid solution. However, if there are several solutions, only those within the values specified by the range facet (if any) are selected as valid solutions. When the information in the range facet is not sufficient to choose a correct solution from multiple solutions, additional or alternative procedure for filtering correct solutions is to use the result of the numeric simulation of its corresponding differential model, as we will show below. A correct solution of an algebraic model describing a behavior in a steady state must agree with the numeric simulation result of its corresponding differential model. This procedure of validating the solution of an algebraic model against the numeric simulation of a differential model has not been automated in the current implementation of ORACLE, and the user should try both models to compare their results.

The previous example showed how ORACLE composes a model to compute the sailboat speed in the equilibrium state of forces. If we are interested not only in such a speed but also in how the boat arrives at the speed, starting from zero speed, the boat speed must be computed as a function of time. Relevant model fragments are retrieved and instantiated in a similar way. In this case, however, the net force on the sailboat is not necessarily zero all the time because the boat accelerates until it reaches the equilibrium state of forces. Therefore, the right hand side of the equation of Newton2 does not become zero, but stays as $\mathrm{d}(\mathrm{p}(\mathrm{t}))/\mathrm{dt}$. Since $\mathrm{p}(\mathrm{t}) = \mathrm{d}(\mathrm{m} \cdot \mathrm{v}(\mathrm{t}))/\mathrm{dt}$, ORACLE solves the differential equation, $\mathrm{F}(\mathrm{t}) = \mathrm{d}(\mathrm{m} \cdot \mathrm{v}(\mathrm{t}))/\mathrm{dt}$ for $\mathrm{v}(\mathrm{t})$ by numeric simulation. A plot of the simulation result in Figure 8 shows that the sailboat ultimately accelerates to the same speed as the one predicted by the algebraic method, thus confirming the algebraic solution. Also notice that the model fragment Newton2 used for modeling the spring-block systems is reused for modeling the sailboat and that model fragments Lift and FDrag are shared by hull and sail.

Issues in Scaling Up

There are several problems raised in scaling up ORACLE not only to broaden the types of physical systems to be modeled but also to model a physical system with a large number of components and phenomena involved.

First, the size of a model generated by the current version of ORACLE can restrict scaling up either because of practical limitations of solving a huge model or because solving a huge model takes too much time to be useful. Table 1 shows the sizes of the models and the times for formulating and solving the models for the examples shown in this paper. The model sizes of spring-block systems with different configurations from the examples in Table 1 are about the same as those of the spring-block systems with the same number of blocks and springs in Table 1, that is, the model sizes of spring-block systems are independent of their configurations. As we can see in Table 1, the size of a model is not directly proportional to the number of model fragments instantiated. Rather it is proportional to the number of variables specified in the problem statement, or to the number of unknowns in the equations of the model (#unknowns includes #variables and newly generated variables in the equations).

As described earlier, a model contains a set of variables, a set of assumptions, a set of names of model fragments, and a list of equations. The equations of a model are composition of the equations of the model fragments instantiated to construct the model. Although the equations of most model fragments exist in a very short, simple form *before* instantiation, they may become very long and complex *after* they are instantiated for the particular entities and physical phenomena in the problem. This explains the large variations in the sizes of models with similar number of model fragments; the big size of a model is attributed to the long equations of instantiated model fragments. Even for a same model fragment, the equations after instantiation can be very different in their sizes depending on for which variables they are to be solved and which variables of the equations are known.

Figure 9 shows the growth rate of the model size and the growth rate of the time for modeling and simula-

example	#variables model fragments (#instantiation) model_gen time	#unknowns model_sol time	#model fragments display_save time	#lines of model total time
algebraic	1	3	5	30
model	FDrag (2), Lift (2), Newton2 (1)			
of boat	4.900	4.616	4.917	14.333
differential	1	3	6	31
model	Newton2 (1), FDrag (2), Lift (2), Lift_at_zero_speed (1)			
of boat	4.700	18.700	4.716	28.116
one	12	12	3	206
block	Newton2 (1), Euler (1), Springforce2 (1)			
system	14.583	19.800	14.833	49.216
two	24	24	6	2010
block	Newton2 (2), Euler (2), Springforce1 (1), Springforce2 (1)			
system	108.533	118.283	110.433	337.249
three	36	36	10	5330
block	Newton2 (3), Euler (3), Springforce1 (2), Springforce2 (2)			
system	339.350	264.616	344.767	948.733

Table 1: Size of a model and time for formulating and solving it for each of the ORACLE examples. #variables is the number of variables in component form, specified in the problem statement; #unknowns includes #variables and newly generated variables in the equations; #model fragments is the total number of model fragments instantiated for the model; #lines of model is a count of lines of the model; model_gen time is CPU time for generating a model; model_sol time is CPU time for solving the equations of a model; display_save time is CPU time for displaying the result of solving the model and saving all the results; the units of the CPU times are in seconds.

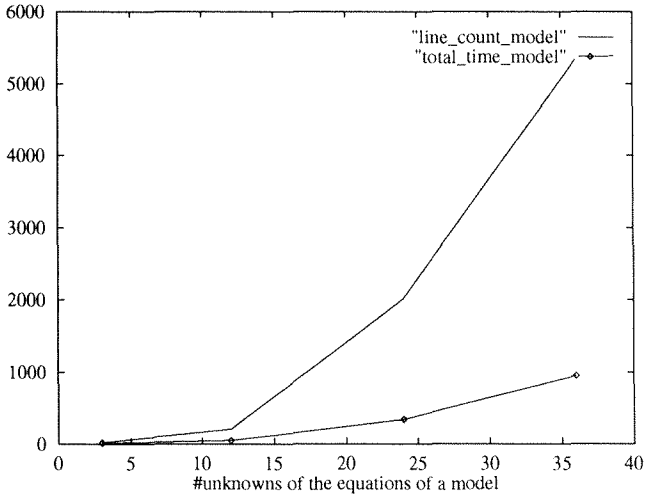

Figure 9: The size of a model and the time for formulating and executing the model. For the count of lines and times of a model with 3 unknowns, the average values of the 2 models with 3 unknowns (algebraic model of boat, differential model of boat) are used. The units of the model sizes are in the line counts of models and the units of the times are in seconds.

tion as a function of the number of unknowns of the a model. The size of a model is measured in terms of the number of lines of the model. When the number of unknowns increases from 3 to 12, the model size increases from 30.05 lines to 206 lines, which is almost 7 times. This is because the spring-block example involves much more complex computation which results in long equations. It is also notable that when the number of unknowns increases from 12 to 24, the model size increases from 206 lines to 2010 lines, about 10 times, and that when the number of unknowns increases from 24 to 36, the model size increases from 2010 lines to 5330 lines. Given limited data, the growth rate of a model is roughly quadratic in the number of unknowns. The total time for generating a model and solving it is also proportional to the number of unknowns of the model, but the growth rate of time is not as fast as that of the model size, as shown in Figure 9.

Some of the large models have long equations which cannot be simplified further in their nature unless we decide to produce approximate models instead. However, some models may be simplified without losing accuracy of their predictions by doing additional processing on the equations instead of applying Maple-builtin simplification functions. Reformulating the model generation and solving process of ORACLE is another direction to consider in order to efficiently construct and solve a large model. Restricting ORACLE to a certain type of physical system is another way to scale up the modeling system to handle a complex physical system with a large number of components and

phenomena involved. For example, if we want a special purpose modeling system for spring-block systems only, we can make the modeling system generate a single model that works for any number of blocks. As the number of blocks increases, the model would need more data storage but the model itself does not become larger. This kind of approach to scaling up a modeling system has advantage of being able to model and simulate a complex physical system without the size problem of a generated model, but has disadvantage of losing the breadth of physical systems covered by the modeling system.

Second, selecting relevant model fragments would become a more important issue in scaling up. Including additional properties of an object, such as electrical or thermal properties, in the description of an entity as slots does not cause a problem in selecting model fragments because when ORACLE determines the relevance of a model fragment to a given problem it checks whether each variable of the model fragment is mentioned as an entity property or variable of the problem statement or derivable from them. The difficulty is in selecting a model fragment among multiple model fragments with same or similar variables but with different assumptions. In the current implementation, model fragments are indexed by the mf slots of entities, with more frequently relevant ones first, and therefore the search process is sensitive to the order of the model fragments. We may want a more efficient method for organizing model fragments which will facilitate identifying and retrieving relevant model fragments.

The third problem in scaling up is related to the size of a generated model. As a model gets bigger, it would become more difficult to understand the model or to validate its solution. Having ORACLE provide an explanation of its solution, or checking the solution against that of an approximate model or experimental data will help understand or validate the model.

Related Work

Falkenhainer and Forbus (1991) describe a form of compositional modeling where a device model is automatically formulated by composing a set of relevant model fragments which are initially obtained by matching the terms of a query to a domain theory and then elaborated later. There are several differences between our work and theirs. In ORACLE we distinguish model fragments from entities; model fragments are used for describing physical phenomena and entities for objects (both composite and primitive); model fragments are indexed by the "mf" slot of an entity. In Falkenhainer and Forbus' approach, model fragments are used for describing all the phenomena, objects, and devices, and are organized into mutually exclusive sets called assumption classes. When the class condition holds, one and only one of the assumptions associated with the class must hold and the model fragment containing that assumption must be included in a model. Once the model fragments with appropriate assumptions are selected, the process of instantiating the model fragments and assembling them is straightforward. While a composite object in ORACLE can consist of any heterogeneous parts, a unique minimal covering of parts taken from a single part-of hierarchy is required to exist in their approach to generate a simplest possible model. They do not have a capability of handling detailed structural relations among parts and choosing appropriate reference frames for parts, and therefore cannot handle complex motions such as motion of multiple objects. For behavior generation Falkenhainer and Forbus use either qualitative simulation by QPE (Forbus 1990) or quantitative simulation by numeric simulation, whereas we use numeric simulation or analytic method.

In (Nayak 1992), Nayak describes a method to construct a device model by selecting an appropriate model for each component of the device using structural, behavioral, and expected behavioral constraints. In his system, a model is formulated by composing a set of model fragments, as in ours. However, the uses of the models produced by the two systems are different. While ORACLE constructs a model to predict motions of physical systems, his system builds a model to explain causal relations between parameters of a device. Another difference is that he uses order of magnitude reasoning for behavior generation while we use numeric simulation. His order of magnitude reasoning method is restricted to generating the behavior at a fixed point in time, but we can predict the behavior changing with time as well as the behavior at a fixed point in time.

The SIGMA system developed at NASA Ames Research Center (Keller and Rimon 1992) is a tool which aids a scientist-user in building a model. After the interaction with the user, it produces a model specified in data flow graph and executes the model to compute a unknown quantity. Like ORACLE, SIGMA organizes and represents domain knowledge in frame structures. However, it is a user-assistant system rather than an autonomous model-building system, and has several restrictions in constructing and executing a model, which ORACLE does not have. For example, multiple quantities cannot be computed at the same time because it cannot solve more than one equations simultaneously, and the types of equations are restricted to algebraic equations or first-order ordinary differential equations; model fragments cannot be put together in an arbitrary order due to the strict backchaining control strategy of its model building process. It converts the input values into a common, consistent set of scientific units, but does not have a provision to transform a vector quantity measured in one reference frame to another, which is necessary in dealing with moving objects.

The MSG system developed by Ling *et al.* (1993) generates mathematical models for analyzing heat transfer behavior. The approach of the MSG system to building a model is similar to that of ORACLE in the

sense that it is compositional. However, there are several differences. While ORACLE focuses on modeling physical systems involving motion, MSG models physical systems involving heat flow. Therefore, the domain knowledge the two systems use are different. A second difference is that ORACLE represents the knowledge explicitly in general, declarative form, but much of the knowledge that the MSG system uses is embedded in the system as part of its algorithm. A third difference is that, while ORACLE generates a model and then solves the equations of the model to predict the behavior of a given physical system, MSG presently does not solve the equations of its generated model.

Another relevant line of work concerns model selection (Addanki *et al.* 1991; Weld 1992), or model simplification (Yip 1993; Falkenhainer 1993), rather than model generation.

Yet another related works concern simulation generation instead of model generation. The SIMLAB system (Palmer and Cremer 1991) produces a simulator from a user-provided physics model. Given a mathematical model of a physical phenomenon and instructions for solving the resulting equations, SIMLAB transforms the model into an executable simulation code to analyze the phenomenon. However, the user still has the burden of creating the mathematical model. The program built by Berkooz *et al.* (1992) is similar to SIMLAB. It is basically a compiler for translating differential equations expressed in mathematical and programming constructs into an executable code. The SINAPSE system (Kant 1992) also automatically transforms a given model into a program in desired language, though again the human user must create the input model.

A number of mechanical device simulators are commercially available, such as ADAMS (Dawson 1985), and DADS (Haug 1989). These programs, like most simulators, incorporate physics knowledge such as Newton's laws of motion directly into algorithms rather than representing them explicitly. The simulators include powerful algorithms for forming and solving the equations of motions for a wide variety of mechanisms, but lack the flexibility that ORACLE has to explicitly instantiate general model fragments in particular situations.

Previous AI research in spatial reasoning about mechanical devices (Faltings 1987; Gelsey 1989; 1994; Joskowicz and Sacks 1991) has devoted considerable attention to reasoning about contacts between solid bodies, a problem ORACLE does not presently address. Like the commercial simulators, these programs incorporate knowledge of physical phenomena directly into algorithms rather than attempting to explicitly instantiate general model fragments in particular situations, as ORACLE does.

Future Work

There are several directions in which the work described in this paper can be extended. In the current implementation of ORACLE, there are only certain classes of entities and model fragments available in the knowledge base. Adding more model fragments and entities would expand the types of physical systems covered by ORACLE. It would also be a valuable test for the extensibility of the system.

Problems which do not involve new physical phenomena, but require model fragments with different assumptions or representations will involve minor changes. For example, the two model fragments for spring forces presently assume a linear spring with linear damping. To model nonlinear springs, we need to add a new model fragment with different equations and assumptions. The types of springs should be considered when an if_added procedure chooses between the two model fragments of spring forces.

Some spatial reasoning problems can be solved by qualitative interpretation of the quantitative models produced by ORACLE. For example, qualitative description of motions (such as translational, rotational, oscillatory, or tumbling) can be easily obtained by postprocessing the simulation results of the models. Coverage of space of a moving object, any regularity of the coverage over time (such as monotonically decreasing coverage of a damped spring), or possible contact/collision with other moving objects (intersection of the coverages during same time intervals) can also be produced by postprocessing the simulation results.

When something goes wrong during problem solving (e.g., a model cannot be solved due to fewer equations than unknowns), ORACLE currently prints the dead-end situation, asks for further information, and quits. The user has to figure out the cause of the problem and rerun the program with new information. In order for ORACLE to suggest possible directions to fix the problem, it must have a capability of reasoning about equations and unknowns.

Conclusion

The model of general motion in three dimension is difficult to formulate by hand but important because of its relevance to many practical applications, including computer graphics, robotics, and design. We have made important progress in automating the model-building process for physical systems with multiple moving objects in arbitrary configurations by developing a new method which uses basic domain knowledge. The method has been implemented in a working program called ORACLE and tested in the domains of mechanical devices and sailboats. Given a description of a problem involving a moving physical system, ORACLE automatically identifies relevant model fragments, instantiates them for the particular entities and physical phenomena in the problem, composes the instantiated fragments to form a model, and simulates the model to solve the problem. Knowledge of physical phenomena is represented with model fragments which can be shared and reused by many models. Most of

the knowledge is just the same fundamental equations that appear in any standard mechanics textbook, with their implied semantics of vectors and frames of references. Starting with the most basic, simple concepts in the domain of mechanics, ORACLE can still generate a powerful model for complex motions. This is a new method which solves several problems with existing AI modeling work on motion by: (1) explicit handling of vector quantities and frames of reference; (2) simultaneous handling of multiple equations (algebraic or differential, linear or nonlinear); and (3) declarative, algorithm-neutral representation of physics knowledge.

As discussed earlier, there are many programs developed for reasoning about motion of mechanical devices. However, the programs do not solve problems in a general method from basic physics principles, but rely on specific methods specialized for certain classes of problems. Letcher (1976), for example, uses a numerical procedure adapted from Newton-Raphson iteration to find the optimum sailboat velocity with the maximum component in the wind direction, or the sailboat velocity which will balance all the forces. ORACLE solves the same sailboat problems without requiring special-purpose problem solving methods, as it does for other mechanical devices.

References

S. Addanki, R. Cremonini, and J. S. Penberty. Graphs of Models. *Artificial Intelligence*, 51:145–177, 1991.

G. Berkooz, P. Chew, J. Cremer, R. Palmer, and R. Zippel. Generating spectral method solvers for partial differential equations. Technical Report 92-1308, Cornell University, 1992.

B. W. Char, K. O. Geddes, G. H. Gonnet, B. L. Leong, M. B. Monagan, and S. M. Watt. *MapleV Language Reference Manual.* Springer-Verlag, New York, 1991.

G. Dawson. The Dynamic Duo: Dram and Adams. *Computers in Mechanical Engineering*, March 1985.

B. Falkenhainer and K. D. Forbus. Compositional modeling: finding the right model for the job. *Artificial Intelligence*, 51:95–143, 1991.

B. Falkenhainer. Ideal physical systems. In *Proc. of the 11th National Conference on Artificial Intelligence*, pages 600–605, 1993.

B. Faltings. *Qualitative Place Vocabularies For Mechanisms in Configuration Space.* PhD thesis, University of Illinois at Urbana-Champaign, July 1987.

K. D. Forbus. Qualitative Process Theory. *Artificial Intelligence*, 24:85–168, 1984.

K. D. Forbus. The Qualitative Process Engine. In D. S. Weld and J. de Kleer, editors, *Readings in Qualitative Reasoning about Physical Systems*, pages 220 – 235. Morgan Kaufmann, 1990.

A. Gelsey. Automated Physical Modeling. In *Proc. of the 11th International Joint Conference on Artificial Intelligence*, pages 1225–1230, 1989.

A. Gelsey. Automated reasoning about machines. *Artificial Intelligence*, 1994. to appear.

G. Hartquist. Public PADL-2. *IEEE Computer Graphics and Applications*, pages 30–31, October 1983.

E. J. Haug. *Computer Aided Kinematics and Dynamics of Mechanical Systems, Volume 1: Basic Methods.* Allyn and Bacon, Boston, etc., 1989.

L. Joskowicz and E. P. Sacks. Computational Kinematics. *Artificial Intelligence*, 51:381 – 416, 1991.

E. Kant. Code synthesis for mathematical modeling. In *Working Notes of AAAI Fall Symposium on Intelligent Scientific Computation*, pages 54–59, 1992.

R. M. Keller and M. Rimon. A Knowledge-based Software Development Environment for Scient ific Model-Building. In *Proc. of the 7th Knowledge-Based Software Engineering Conference*, 1992.

B. Kuipers. Qualitative Simulation. *Artificial Intelligence*, 29:289–388, 1986.

J. S. Letcher, Jr. Optimum windward performance of sailing craft. *Journal of Hydronautics*, 10:140–144, 1976.

S. R. Ling, L. Steinberg, and Y. Jaluria. MSG: A Computer System for Automated Modeling of Heat Transfer. *Artificial Intelligence for Engineering Design, Analysis and manufacturing*, 7(4):287–300, 1993.

M. Minsky. A Framework for Representing Knowledge. In *The Psychology of Computer Vision*, pages 211 – 277. McGraw-Hill, New York, 1975.

P. P. Nayak. *Automated Modeling of Physical Systems.* PhD thesis, Stanford University, 1992. STAN-CS-92-1443.

R. S. Palmer and J. F. Cremer. SIMLAB: Automatically creating physical systems. Technical Report 91-1246, Cornell University, 1991.

P. Struss. Global Filters for Qualitative Behaviors. In *Proc. of the 7th National Conference on Artificial Intelligence*, pages 275 – 279, 1988.

D. S. Weld. Comparative Analysis. *Artificial Intelligence*, 36:333–374, 1988.

D. S. Weld. Reasoning about model accuracy. *Artificial Intelligence*, 56:255–300, 1992.

B. C. Williams. Doing Time: Putting Qualitative Reasoning on Firmer Ground. In *Proc. of the 5th National Conference on Artificial Intelligence*, pages 105 – 112, 1986.

K. M. Yip. Model Simplification by Asymptotic Order of Magnitude Reasoning. In *Proc. of the 11th National Conference on Artificial Intelligence*, pages 634–640, 1993.