AUTOMATIC DERIVATION OF THE EQUATION OF MOTION OF A PENDULUM

MICHAEL BEESON

Abstract. Some knowledge of elementary physics has been formalized in first-order logic. The domain of discourse includes physical objects and their relations, mathematical formulas, and the semantic relation between formulas and objects. The knowledge in question has been written in Prolog and is sufficient to support an automatic derivation of the differential equation of motion of a pendulum. The inference engine makes use of the Knuth-Bendix method and also of a symbolic computation system for algebra and calculus. Perhaps this is the first program to use both knowledge representation in logic and symbolic computation.

1. Introduction

In physics textbooks one finds "derivations" of differential equations of motion describing various physical systems, for example a pendulum. As an exercise in both knowledge representation and automated deduction, we have developed methods of formalizing and automating such derivations.

The domain of discourse in such discussions cannot be restricted to physical objects and their properties and relations though of course it must contains such concepts as *mass*, *force*, *inertia*, *time*, and *length*. It must in addition be able to formalize such expressions as "the velocity of the pendulum bob at time t is given by $f(t)$." Thus one of the types of objects to be considered is the type *mathematical expression*, and there will be relations of the form "expression E denotes R", where R is a physical object or relation.

From the logical point of view, we are involved in formalizing a metalanguage. Ordinary mathematics can be thought of as the "object language", which talks about physical reality. What concerns us here is the semantic relationships between this language and physical reality: in what sense can it be formally said that a certain differential equation is the equation of motion of a pendulum? When the physics texts say that "the equation of motion of a pendulum is derived", they mean more precisely that it is demonstrated that a certain relationship holds between that equation (as a sequence of symbols) and physical pendular.

In the physics textbooks, however, most of the steps of the derivation are ordinary mathematical transformations on equations. These actually correspond to inferences on the meta–level: suppose we have proved that equation E describes the physical situation

Date: This was presented to the Stanford Artificial Intelligence Seminar in April, 1987, and retyped in 2012 with only grammatical editing.

Then we transform E to a mathematically equivalent equation H. We may then infer that H describes the physical situation.

Since the bulk of the reasoning (as printed in textbooks) is ordinary mathematics, it is clear that purely logical reasoning will not suffice to automate these derivations. We will need a theorem prover operating at the meta–level, coupled to a system for ordinary mathematical calculations.

There has been much work in artificial intelligence directed towards "qualitative physics"; see e. g. Bobrow [1]. Our work, by contrast, deals with the formalization of "quantitative physics." We think that before trying to formalize common sense notions of physical objects and their behavior, we ought to get some practice by formalizing the precise and mathematically elegant notions of Newtonian mechanics.

This work is directed towards answering the following questions:

- *What knowledge structures are needed for the logical representation of Newtonian mechanics?*
- *What knowledge structures are needed for the representation of the semantic relations between physical reality and the equations of Newtonian mechanics?*
- *How can a theorem prover be designed that calls on an efficient symbolic computation system when ordinary mathematical computation is needed?*

As described above, our work is a purely theoretical scientific endeavor. Yet it would be misleading to leave matters there, for what first attracted our attention to the problem was a practical matter. We have been involved in applications of artificial intelligence to computer assisted instruction, and we were thinking about what would be involved in designing intelligent computer programs to teach physics. It seems desirable to have a system that would permit the user to describe a physical system, perhaps by graphically "building" an image of the system by selected and moving components such as weights, rods, and hooks. The computer should then be able to automatically generate a simulation of that physical system, numerically solve the differential equations of motion, and use the solution to drive the graphics that would present to the user a model of the system. (Credit for the idea of a dynamic simulation generator belongs to Henson Graves.) We have seen many examples of particular simulations, each one specially coded, but nothing like what Graves envisions has been built [still true as this is re-typed in 2012, as far as I know]. The theoretical work described in this paper grew out of these considerations.

2. Knowledge representation: explicit use of the meta–level

Reflection reveals that every use of mathematics in science involves the notion of the physical interpretation of the mathematical symbols. Hence any system for formalizing scientific arguments must involve the explicit representation of meta–level knowledge. The lack of meta–leval representation has been recognized as a limitation of current expert systems; see Genesereth [2] for a description of an attempt to incorporate meta-reasoning in expert system technology.

We formalized and automated the derivation of the equation of motion of the pendulum. But it is not even possible to state what it means to be the equation of motion of the pendulum, let alone state the conditions on the pendulum under which the equation describes its motion, without using knowledge about physical quantities and the interpretation of mathematical symbols in the physical world. To be explicit, our system has a threefold domain of discourse:

- *Physical objects and their relations, such as the mass or location of an object, or the force of one object on another.*
- *Mathematical equations and their manipulation, including the derivation of new equations from old ones, and the evaluation of expressions by algebra or calculus.*
- *The semantic relations or "modeling functions" by which an expression or equation denotes an object or relation.*

One may view this domain of discourse as an extension of the familiar situation in analytic geometry, in which numbers describe space by means of a certain "modeling function." We usually think of this process by the *inverse* of the modeling function, the *coordinate functions*. One may say that mathematical equations of motion "coordinate" dynamics in the same way that Cartesian coordinates "coordinatize" statics.

The remainder of this section is devoted to a description of the logical representation of the knowledge needed to formalize the derivation of the equation of motion of a pendulum.

We have a unary predicate $object(a)$, intended to denote physical objects such as apples, pendulum bobs, or the earth. We shall usually use letters near the beginning of the alphabet for objects. We shall also need the concept of *physical point*. We shall usually use letters p or q for physical points. We introduce the unary predicate *point*?. Note that a point is not the same as a number or vector. A point gives rise to a vector when an origin and a system of units is specified. Once we have a vector we still do not have numbers until coordinate axes are specified; then there are three functions $xc(v)$, $yc(v)$, and $zc(v)$ that get the spatial coordinates of a vector v. When the origin of time is specified we also have $tc(v)$. The values of the coordinate functions are real numbers; we shall usually use letters near the end of the alphabet for real numbers. The careful distinctions between points, vectors, and (triples of) numbers are essential for the formalization.

We need a binary predicate $on(p, a)$ to express that point p is on object a. Two objects may be connected; this is expressed by the binary predicate $connected(a, b)$. It is a symmetric relation and if a is connected to b then they have at least one point in common; the point $pointOfConnection(a, b)$ is on both a and b. To deal with the example of pendula, we will not have to worry about the various complicated ways in which two objects can be connected.

An object has a location and a mass. The location of an object is a region of space. We may idealize and consider the case of a "point object", whose location is a point. In the case of a point object, $position(a)$ is synonymous with $location(a)$. Its value is of type *point*. If coordinates have been introduced, we may write $xc(location(a))$ as $xc(a)$, since this latter expression can have no other interpretation. Thus the coordinate functions can apple either to points or to vectors.

We have a binary function symbol *force*(a, b), intended to denote the force of object a on object b. However, the arguments of *force* are not restricted to be physical objects; we can also write *force*$(gravity, b)$ for the force of gravity on object b. We shall not need in this paper any electromagnetic or other kinds of forces, but they could fit into the same formalize. The value type of the expression *force*(a, b) is *vector*. Note that *force*(a, b) does not denote a number.

How do we get vectors from points, or numbers from vectors? We need a function *vector*(a) that gets the vector associated with the point a, and the function *value*(v) that represents the member of $\mathbb{R}^3$ (triple of numbers) associated with the vector v. Both of these functions depend on other things: *vector* depends on the choice of units and origin, and *value* in addition depends on the choice of axes. In order to formalize this we need either a system with very rich types (some generalization of typed λ-calculus), so that we can represent the function that leads from coordinate ridings and systems of units to the function *value*, or else we need a completely type-free system such as first-order logic. We have been able to formalize this satisfactory in Prolog by the simple device of allowing function symbols to take a variable number of arguments Thus we write $value(a, Units, Origin)$ or just $value(a)$ as the need requires. The composition $value(vector(a))$ is the triple $[xc(a), yc(a), zc(a)]$.

Similarly, the location of a point may depend on time. In Prolog we simply use *location* as a function of either one or two variables, writing *location*(a, t) for the location of object a at time t. In typed λ-calculus we could formalize the function that passes from t to the function $\lambda a\, .location(a, t)$, i.e. to the unary *location* function.

We want to write *distance*(a, t) for the distance between two points, but of course that can only be defined relative to a system of units. It is, however, independent of the choice of origin of coordinates or of axes. We formalize this in logic by using *distance* as a three-place function *distance*$(a, b, units)$, or as a binary function if the units are fixed. The relation with Euclidean distance is given by the equation

$$distance(a, b, units) = \ |value(a, units, Origin) - value(b, units, Origin)|$$

where *Origin* is a free variable.

Points may have *velocity*. Velocity is a fundamental physical concept, as is acceleration. What is the correct formal sense of the informal statement that "velocity is the time derivative of position"? Only mathematical expressions denoting number–valued or vector–valued functions can be differentiated; what must be meant is that if expression E denotes *position*(a), then dE/dt denotes *velocity*(a). Examine the types carefully here: *position*(a) is a *point*, and E and dE/dt are *expressions*. When we say "E denotes *position*(a)", we mean that the vector determined by E (with a given assignment of values to free variables) is the same triple of numbers as *value*(*position*(a)).

Similarly, the physical quantity *acceleration* is denoted by the time derivative of the expression denoting the velocity. Note the abbreviation "time derivative" for "derivative with respect to the variable denoting time." The Prolog code makes the distinction explicitly and declares that 't' will denote time unless otherwise specified.

At this point enough knowledge has been introduced to formalize Newton's laws. Newton's first law serves to evaluate terms of the form $f(a, b)$, by specifying "To every action there is an equal and opposite reaction":

$$force(a, b) = -force(b, a).$$

Newton's second law asserts that $F(b) = \text{mass}(b)acceleration(b)$ is a valid equation in the strict semantic sense that its left and right sides denote the same vector, where $F(b)$ denotes the resultant of all forces on an object b. That information can be used to derive other valid equations, or to conclude that the denotations of the two sides of the original equation are the same. Of course, we can first derive other valid equations and then conclude that their two sides have the same denotation.

Finally, Newton's third law of universal gravitation specifies what the gravitational forces of objects on each other are. Let us define $F(a, b)$ to be the magnitude of the force of a on b:

$$F(a, b) = \ |value(force(a, b))|$$

Then Newton's third law is

$$F(a, b, gravity) = \frac{-Gmass(a)mass(b)}{distance(a, b)^2}$$

A well-known argument shows that if $on(location(b), surfaceOfEarth)$, then

$$F(earth, b, gravity) = \ -gmass(b).$$

In this situation we abbreviate $F(earth, b, gravity)$ to $F(gravity, b)$. Although there are problems of approximate equality involved in using the equation for objects whose location is only near the surface of the earth, we shall assume the usual law for the force of gravity, in which **down** is a unit vector directed at the center of the earth:

$$force(\ gravity, b) = \ -g\mathbf{down}mass(b).$$

Physics books often contain problems about finding the value of g at the top of Mount Everest. The problems of approximate equality can be solved in the same way as the problems of *value*, by regarding g either as a function of one argument (height above the surface of the earth) or if that is regarded as fixed, as a function of zero arguments, i.e. a constant.

In order to analyze the pendulum, we assume that only mechanical and gravitational forces are acting. That is expressed by saying that the total force on an object c is given by the sum of $force(gravity, c)$ and terms $force(a, c)$ summed over all objects a which are connected to c. This assumption is easily expressed in our formal language (and in Prolog), although it should be noted that is mathematical expression involves a summation whose index involves a logical condition using the physical relation *connected*.

Specializing now to the pendulum problem, we define the data type *pendulum*: a pendulum is a triple $[a, b, c]$, whose members are objects, with a connected to b and b connected to c. We call a the "pivot", b the "rod", and c the "bob." Moreover, if $[a, b, c]$ is to qualify as a pendulum, nothing but b must be connected to c and nothing but a and c must be connected to b.

That definition is very general. We shall be concerned with "simple pendula." A pendulum is called *simple* provided:

(1) The pivot a is *fixed*, i.e. *location*(a) is independent of t. This can be expressed by requiring *location*$(a, t_1) =$ *location*(a, t_2). We must also require
$$\textit{location}(a) = \textit{pointOfConnection}(a, b)$$
to rule out pendular attached to an immovable point a by a moving pivot.
(2) The rod is massless: *mass*$(b) = 0$.
(3) The rod is rigid: Let L be the distance between *pointOfConnection*(a, b) and *pointOfConnection*(b, c). Then L is independent of time.
(4) The bob is a point mass: *location*$(c) =$ *pointOfConnection*(b, c).

Note that a simple pendulum can still be quite complicated: there can be frictional forces at the connections, and the amplitude of motion can be arbitrary, including even "going over the top."

We want to consider only frictionless pendula. As it turns out, the assumption that the pendulum is frictionless enters into the derivation at only one point: when one wishes to prove that the force of the rod on the bob is directed along the rod, i.e. has no tangential component. This can be expressed in symbols by

$$\textit{force}(b, c) \cdot \textit{velocity}(c) = 0. \tag{5}$$

In all the physics books I examined, this point of the derivation is passed over in a line or even between the lines. It is actually quite difficult to derive this equation from first principles about friction. One might argue intuitively that in a frictionless pendulum, one could as well replace the rod by a string, and that if the force of rod on bob were not radial, the string would bend. Of course this argument has no rigorous value. The attempt at formalization has turned up this flaw in the derivations in all the textbooks!

Note that, physically speaking, there may be only one direction in which the z-axis can be placed to make (5) valid: if the pivot-to-rod connection is essentially one-dimensional, we will have to place z so that the motion will be in the x-y plane. Assuming the validity of (5) with respect to *all* coordinate systems amounts to assuming there is a frictionless universal joint. Strictly speaking, (5) does state the equality for all coordinate systems, since it is an equation between vectors. Our derivations, however, will be valid also for pendula that satisfy (5) only when projected on the x-y plane, i.e. satisfy the two equations obtained by dotting the values of the sides of (5) with $(1, 0, 0)$ and $(0, 1, 0)$.

Since our original aim was to formalize the derivations in the textbooks, we shall simply assume (5) as an additional hypothesis. We define a pendulum to be *frictionless* if (5) holds. Choose coordinates so that the origin is at *location*(a), the y axis is directed upward, and the z axis is arbitrary. By "upward" we mean that we assume

$$xc(\mathbf{down}) = 0, \quad yc(\mathbf{down}) = -1. \tag{6}$$

Let θ be the angle between the bob and the downward vertical, i.e.

$$\theta(t) = \arctan\left(\frac{xc(\textit{location}(c, t))}{-yc(\textit{location}(c, t))}\right) \tag{7}$$

Introduce the customary notation $\dot{x}$ for dx/dt and $\ddot{x}$ for d^2x/dt^2. The equation of motion of a simple frictionless pendulum, whose validity is the main concern, can be written as

$$\ddot{\theta} = -\frac{g}{L}\sin\theta \tag{8}$$

The main theorem, for which we have given a formal and automatic demonstration, is that equation (8) is valid, under the stated assumptions (1) through (7). The next section discussed the derivation.

3. Inference Technique

Prolog and symbolic computation. Our idea was to combine the built-in inference mechanisms of Prolog with the symbolic manipulation capabilities of another program. We have such a program at hand, written in Prolog by the author for another project. [Note added in 2012: this was an early version of MathXpert.]

The reflection principle. Suppose we have derived *valid*($e1 = e2$), where $e1$ and $e2$ are expressions represented as Prolog terms; equality is just another function symbol to Prolog. Suppose we pass the expression $e1 = e2$ to the symbolic manipulation routines, which rewrite it, deriving the expression $e3 = e4$. Then we will permit the inference *valid*($e3 = e4$). This inference rule we call the "reflection principle", after similar principles that have been studied in mathematical logic. It "reflects" object–level derivations (computations) into meta–level derivations.

The Knuth–Bendix method. A considerable body of literature has developed about the use of rewrite rules in automated deduction, beginning perhaps with the often cited paper [3]. The central idea of this method is that if a certain expression can be rewritten in two essentially different ways, then one may deduce a new equation by setting the two answers equal. Suppose $e1$ can be rewritten as $e2$ and also as $e3$. Then the reflection principle allows us to conclude *valid*($e2 = e3$). This is the main way that the reflection principle is applied in our derivations.

Our original proofs proceeded directly in the Knuth–Bendix style, using rewrite rules for algebraic manipulation. However, as is well known in the field of symbolic computation, this is inefficient. We therefore make use of more efficient routines for algebraic and calculus computations.

Creativity in deriving new results using this method lies solely in choosing the proper starting point or points. Once the starting expression is supplied, it is simply a matter of simplifying it in two different ways to obtain the equation of motion of the pendulum, using for the simplification two kinds of steps: steps using the axioms (1) through (7), and steps using symbolic computation and the reflection principle.

Suppose $[a, b, c]$ is a simple frictionless pendulum. We let v be the velocity of c. We start with the express $F \cdot v$, where $\cdot$ means vector dot product, and F is the sum of the forces on c. The rules of physics and mathematics (1) through (7) are sufficient to rewrite this expression in two different ways (automatically): on the one hand F can be rewritten as *force*(b, c) + *force*(*gravity*(c), and on the other as $m[\ddot{x}, \ddot{y}]$, where m denotes the mass of c and x and y are the coordinates of the point *location*(c). Note that Newton's second law

was used for this rewriting. Now the equation (5) implies that $force(b, c) \cdot v = 0$. $F \cdot v$ can be rewritten on the one hand as $force(gravity, c) \cdot v$, which can be further rewritten as $mg \cdot \mathbf{down}$. On the other hand $F \cdot v$ can be rewritten as

$$m[\dot{x} \cdot \ddot{x}, \dot{y} \cdot \ddot{y}].$$

(For simplicity, we work in two space dimensions; the third would drop out anyway in a few lines.) Equating the two rewritings and using (6), we have

$$mg[0, -1] \cdot v = m[\dot{x} \cdot \ddot{x}, \dot{y} \cdot \ddot{y}].$$

This can be recognized (by a human) as the equation of conservation of energy; it says the time derivative of potential energy mgy plus kinetic energy $\frac{1}{2}mv \cdot v$ is zero.

Now the derivation has to be guided for the first time since the initial choice to rewrite $F \cdot v$ in two ways. Namely, we introduce polar coordinates, by the equations

$$\begin{aligned} x &= L \sin \theta \\ y &= -L \cos \theta \end{aligned}$$

If we put these rules in to begin with, the final equation is reached without intervention but the conservation of energy equation does not occur in the derivation. Once polar coordinates are introduced, the symbolic computation system is capable by itself of deducing the final answer, equation (8). The derivation involves more manipulation that one would expect of average freshmen. For the computer, the most difficult part is that the expression must get longer before it gets shorter; the distributive law has to be applied at a certain point, lengthening the expression, before trigonometric identities can shorten it. One may not use the distributive law every time it is applicable without unsavory results, since it makes expressions longer when one doesn't want the longer. A certain look–ahead is prudent: only distribute if there is something to be gained. With that strategy in the program, the derivation of the pendulum equation is well within the computer's grasp.

References

[1] Bobrow, D. B. (ed.) *Qualitative Reasoning about Physical Systems*, second edition, MIT Press, Cambridge, MA (1986).

[2] Genesereth, M. R., an overview of meta–level architecture, *Proceedings AAAI-84*, pp. 119-124.

[3] Knuth, D. E., and Bendix, P. B., Simple word problems in universal algebas:, in: Leech, J. (ed.) *Computational Word Problems in Abstract Algebra*, pp. 263-297, Pergamon Press (1970).