PHYSICS FOR ROBOTS

James G. Schmolze

BBN Laboratories Inc.
10 Moulton Street
Cambridge, MA 02238

ABSTRACT

Robots that plan to perform everyday tasks need knowledge of everyday physics. Physics For Robots (PFR) is a representation of part of everyday physics directed towards this need. It includes general concepts and theories, and it has been applied to tasks in cooking. PFR goes beyond most AI planning representation schemes by including natural processes that the robot can control. It also includes a theory of material composition so robots can identify and reason about physical objects that break apart, come together, mix, or go out of existence. Following on Naive Physics (NP), issues about reasoning mechanisms are temporarily postponed, allowing a focus on the characterization of knowledge. However, PFR departs from NP in two ways: (1) PFR characterizes the robot's capabilities to act and perceive, and (2) PFR replaces the NP goal of developing models of actual common sense knowledge. Instead, PFR includes all and only the knowledge that robots need for planning, which is determined by analyzing proofs showing the effectiveness of robot I/O programs.

1. Introduction

Physics For Robots (PFR) represents knowledge of everyday physics according to the physical capabilities and planning needs of robots. This knowledge is intended to be an important part of the overall knowledge given to a robot. Physical capabilities are represented within PFR by specifying the perceptual and action functionality of a (hypothetical) robot. This specification is comprised by an I/O programming language, whose primitive instructions correspond to primitive perceptions and actions, and an operational semantics, which describes the real world effects of executing I/O programs. (Given the complexity of the real world, this semantics is necessarily incomplete.) The hypothetical robot used for this research has capabilities that are beyond current, but are within near future technology. Some of the robot's capabilities and an I/O program are presented later in this paper.

This research was supported by the Advanced Research Projects Agency of the Department of Defense and was monitored by ONR under Contract Nos. N00014-77-C-0378 and N00014-85-C-0079. The views and conclusions contained in this document are those of the author and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

PFR's representation of everyday physics is very similar in style to Hayes' Naive Physics (NP) formalizations [Hayes 85a, Hayes 85b]. Like NP, PFR focuses on characterizing knowledge while postponing implementation considerations. However, NP is ultimately after realistic models of common sense (see [Hayes 85a], page 5) whereas PFR is after the knowledge that robots need to plan for everyday tasks. As a result, PFR includes a specification of the robot's I/O capabilities whereas NP postpones such considerations. More importantly, PFR includes a criteria for judging the value of its representations whereas NP must rely on the existing, and small, body of what is known about common sense along with one's own intuitions.

One begins to evaluate a PFR representation by selecting a set of everyday tasks for the robot to perform, and for each task, designing an I/O program that, when executed, will cause the robot to successfully perform the task. An I/O program is one whose primitive instructions are only perceptions and actions for the robot to perform (see Section 4). The test for PFR is whether or not its theory of everyday physics is adequate to prove that the execution of each program will accomplish its corresponding task. The more programs/tasks that can be proven correct using a PFR representation, the greater the PFR's expressive power and the better the PFR. Further, given two expressively similar PFR representations, one should choose the simpler of the two, and one should choose the representation that is most in keeping with what is known about common sense.

I point out that there are two notions of correctness here. One is whether or not executing a program will *actually* accomplish the given task in the real world. PFR cannot be used to show this directly. For hypothetical robots, only informal arguments can used here. For actual robots, the programs can be executed and the robots observed. The second notion of correctness corresponds to whether or not executing the program accomplishes the task according to the theories of a PFR representation. The extent to which these two notions of correctness are in agreement is the extent that the representation is successful.

2. Composition of Materials

Physical objects in the everyday world can come into or go out of existence, break apart, come together or mix. Examples from cooking include water that boils and turns to steam, or the pouring of hot water over coffee grounds to create a cup of coffee. PFR must provide the robot with knowledge to deal with such phenomena by giving it a theory of material

composition. Such a theory provides a robot with the skills to:

- identify physical objects as they come into or go out of existence, or go through transformations, and
- determine the properties of whole objects from the properties of their parts, and vice versa, including when the parts are not readily identifiable (such as the portion of the hot water that went into a cup of coffee).

My theory of material composition includes three components: (1) a theory of what constitutes the physical objects, (2) the part-whole relation along with a theory that identifies parts from wholes and vice versa, and (3) a theory that determines the properties of parts from the properties of wholes and vice versa. In this paper, I will only touch on (1) and (2), and will ignore (3) completely given that I will focus on processes. (See [Schmolze 86] for a fuller treatment of material composition.)

Before discussing physical objects, I now introduce some basic elements of PFR. Instants of time are represented as individuals where they form a continuum. Let "seconds" map real numbers to instants where "seconds(n)" denotes n seconds. Points in space form a 3-dimensional continuum. Changing relations are represented as functions on instants of time. Formulas and terms for these relations are written with the time argument separated. For example, "occ.space(x)(t)" denotes the set of points in space that x occupies at time t. "occ.space(x)(t)" is defined iff x is a physical object, t is an instant of time, and x exists at t. Further, x must occupy a non-empty set. Also, "vol(x)(t)" denotes the volume occupied by a physical object at time t, which is defined as the volume of "occ.space(x)(t)", and which is greater than zero for existing physical objects.

A *quantity*, borrowed from Hayes [Hayes 85a], is a set of measurements of a given type. For example, the temperatures and the volumes each form a quantity. Each quantity forms a continuum. I will introduce functions from the reals to various quantities, in the style of Hayes, as needed. For example, "cups(4)" denotes a volume of 4 cups.

Types that are not time-varying are called *basic* types. An example is being a physical object or a temperature. (See [Schmolze 86] for the reasons for the above design choices.)

Regarding notation, boolean function names, i.e., predicate names, will be capitalized. Other function names are written in all lower case. Names of constants are written in all capital letters. Names of variables are written in lower case. Variable names beginning with "t" are implicitly of type "Instant", which denotes the basic type for instants of time. I will write "$(t_1..t_2)$" to denote the open interval from t_1 to t_2. Also, I will use the following shorthand when a time varying predicate, say P, is true over an open interval.

$$[\forall t_1,t_2][P_{(t_1..t_2)} \leftrightarrow [\forall t \in (t_1..t_2)][P(t)]] \quad (1)$$

Being a physical object is a basic type, and I write "Phys.obj(x)" when x is an individual physical object. In order to represent physical objects coming into and going out of existence, I introduce existence as a property of physical objects. Let "Exists(x)(t)" be true when x is a physical object that exists at time t. Physical objects include those objects normally considered as such, e.g., books, cars, computers, the atmosphere, oceans and glasses of water. However, for certain types of transformations that physical objects undergo, it will be useful to include very small physical objects -- possibly objects at the level of atoms and molecules. For example, the process of evaporation can be described by having small pieces of liquid turn to gas and leave the container holding the liquid. Also, by adding some sugar to water and stirring, the entire glass of water becomes sweet. By using small pieces again, one can describe mixtures and show the spread of the sweetness as a dispersion of small pieces of sugar. When hot water is poured over coffee grounds, a new object is created: coffee. It too is a mixture, which can be useful for determining that, say, the coffee is hot because it is primarily composed of pieces of water that were hot just a few seconds earlier.

Hayes ([Hayes 85b], page 74) eschews an atomistic theory because he considers it to be beyond the realm of common sense. In traditional physics, there is a complicated gap to bridge between the microscopic and macroscopic versions of certain properties such as temperature, volume and state. Does the robot need to know about actual atoms and molecules, and if not, what simpler theory will meet the robot's needs?

Fortunately, there is a way to meet the robot's needs without introducing microscopic versions of temperature, volume and state. To this end, I invent a class of physical objects that I call *granules*. Their essential properties are that:

- they are small enough to be a part of all solid, liquid and gaseous physical objects -- they are too small to be seen individually,
- they are large enough to have the usual macroscopic properties of temperature, state and volume (each has a volume greater than zero),
- they are pure, be they purely water, wood, or whatever, and
- they have no proper parts, and consequently, no two granules share parts nor occupied space.

Further, granules of the same type are similar. For example, two water granules with the same heat content will have the same temperature and state. Granules form the smallest physical objects in my ontology. I let "Granule" denote a basic type for granules.

By coupling the part-whole relation with granules, I have a powerful tool for describing material composition. Let "Part(x,y)(t)" be true iff x and y are physical objects that exist at t and x is a part of y at t. "Part" forms a partial order over existing physical objects at each instant. From these relations, I can define a function, called "gset", from physical objects to the sets of granules that comprise them at an instant.

$$[\forall y,t][gset(y)(t) = \{x|Granule(x) \wedge Part(x,y)(t)\}] \quad (2)$$

I will use the ability to determine an object's "gset" as the criteria for identifying the object. For example, let there be a glass called G that contains some liquid at time T. If G and T are identified to the robot, I can identify *the liquid in G* as W with the following.

$$gset(W)(T) = \{x|Granule(x) \wedge Liquid(x)(T) \wedge Contains(G,x)(T)\} \quad (3)$$

"Liquid(x)(t)" is true iff x exists and is entirely liquid at t. ("Solid" and "Gas" are defined similarly for the solid and gaseous states.) "Contains(x,y)(t)" iff x and y exist and x contains y at t. Borrowing from Hayes [Hayes 85b], I have used containment to identify this liquid object.

I can go a step further and write a general rule that allows the robot to identify a contained quantity of liquid as a physical object. The first line in Formula 4 requires that there is *some* liquid in a container and

the remainder asserts the existence of the object formed by all the liquid in the container.

$$[\forall c,t][([\exists x][\text{Contains}(c,x)(t) \wedge \text{Liquid}(x)(t)]) \rightarrow [\exists l][\text{Phys.obj}(l) \wedge \text{Exists}(l)(t) \wedge \text{gset}(l)(t)=\{y|\text{Granule}(y) \wedge \text{Liquid}(y)(t) \wedge \text{Contains}(c,y)(t)\}]] \quad (4)$$

Here, x can be a single liquid granule.

Space does not permit a thorough examination of the utility of granules. The interested reader should refer to [Schmolze 86] where there are rules that allow the robot to identify liquid objects that are poured elsewhere, are mixed with other liquids, partially evaporate, etc. In addition, there are rules that allow the robot to infer various properties of these transformed objects, such as their temperature, volume or composition, all without special knowledge about the properties of microscopic objects. Further, the robot needs to reason about granules only when necessary; it can reason about normal physical objects without considering granules. The general PFR representation thus far allows a wide variety of such rules to be formulated. However, the actual rules for identifying objects to be given to a particular robot will be application dependent.

3. Simple Processes

Any robot that deals with the everyday world must be able to predict changes due to nature. An important source of natural changes is natural processes, and so, PFR includes them. I have limited my study to a class of process types that I call *simple*. All simple process types have an enabling condition and an effect, both of which depend only on the physical condition of the world (and not on, say, the intention of any agent). Basically, an instance of a simple process type occurs when and only when the enabling condition is true for some set of physical objects, and the process has the given effect on the world while it is occurring. For example, whenever a faucet's knob is open, water flows from the faucet. Or, whenever two physical objects are of different temperatures and are in thermal contact, heat flows from the hotter to the cooler object. I note that many real processes are not simple.

Given instances of simple process types (i.e., simple processes), a robot must be able to determine when they occur, how to identify them (e.g., deciding when two processes are the same or different), and what their effects are. Further, these factors must be determinable from limited information. For example, it must be possible to determine a process' effects without knowing when the process will end. Also, the manner of describing effects must allow for either discrete or continuous changes. For example, heat is measured on a continuum, so heat transfer causes continuous changes. However, water flowing from a faucet is (eventually) measured by the transfer of whole water granules, so faucet flow causes discrete changes. Finally, the representation must allow for situations where several processes affect the same property of the very same objects, such as a heating and cooling process occurring simultaneously on the same pot of water.

I note that Hayes [Hayes 85a, Hayes 85b] does not address these points directly. Others, such as [Forbus 85] and [Hendrix 73] have addressed some but not all of them.

I represent simple processes as individuals. Let "Occurs(x)(t)" be true iff x is an event that is occurring at time t. "Occurs" for events is analogous to "Exists" for physical objects.

I will illustrate the essential properties of simple process types by describing the process type for water flowing from a kitchen faucet. Along with that, I will describe faucets, objects associated with faucets (such as their controlling knobs), and their operation. Let "Faucet.flow" be a basic type for faucet flow processes. Each simple process has a set of *players*, i.e., the physical objects that are involved. For "Faucet.flow", the only player is the faucet, with which I associate other objects. In my model, a faucet has a knob, a head, a sink, a supply container that holds the faucet's supply and, of course, the water in the supply container. Let "Kitchen.faucet" and "Faucet.knob" be basic types for kitchen faucets and their controlling knobs, respectively. The knob has fully closed and fully open positions, and there are positions in between. Let "closed.position(k)(t)" denote the space that a faucet knob, k, must occupy in order to be fully closed at time t. Let "open.position(k)(t)" be similar, but for the fully open position. From these functions, I can define "Closed.knob(k)(t)" as true iff k is a faucet knob that is fully closed at t and "Open.knob(k)(t)" as true iff k is a fully open faucet knob at t.

$$[\forall k,t][\text{Closed.knob}(k)(t) \leftrightarrow \text{Faucet.knob}(k) \wedge \text{occ.space}(k)(t)=\text{closed.position}(k)(t)] \wedge [\forall k,t][\text{Open.knob}(k)(t) \leftrightarrow \text{Faucet.knob}(k) \wedge \text{occ.space}(k)(t)=\text{open.position}(k)(t)] \quad (5)$$

If neither is true, the knob is in between. In addition, let "knob.of.faucet(f)(t)", "supply.cont.of.faucet(f)(t)" and "supply.of.faucet(f)(t)" denote the existing knob, supply container and water supply, respectively, of f when f is an existing faucet.

The enabling condition for the "Faucet.flow" process type is written over an interval of time (I will soon explain why) and is true iff a faucet, f, is not fully closed over some open interval, "$(t_1..t_2)$". The following is written with f, t_1 and t_2 free. k is used to simplify the formula.

$$[\forall t \in (t_1..t_2)][\sim\text{Closed.knob}(k)(t)] \quad (6)$$

where "k" is "knob.of.faucet(f)(t)"

I will write "Faucet.not.closed(f)(t_1,t_2)" as a shorthand for Formula 6.

The effect of a "Faucet.flow" process is that water flows from the faucet's supply container to a receiving container, which is either the faucet's sink, or an open container under the faucet's head. To describe the effect, I rely on two defined predicates, "Liq.xfer" and "rate.liq.xfer" (only "rate.liq.xfer" will be formally presented here). "Liq.xfer(c_1,c_2,t_b,t_e)" is true iff the following holds.

1. There is some liquid in a container, c_1, at t_b.
2. Throughout the open time interval from t_b to t_e, where "$t_b<t_e$", granules from the liquid in c_1 are transferred to a different container, c_2. The transfer could have begun before t_b and could have ended after t_e. "Liq.xfer" only states that a transfer occurred throughout the particular interval "$(t_b..t_e)$". Further, the liquid need not remain in c_2 (e.g., it could be transferred elsewhere).

"rate.liq.xfer(c_1,c_2,t_b,t_e)" denotes the average rate of a liquid transfer satisfying "Liq.xfer(c_1,c_2,t_b,t_e)". It is just the volume of the liquid actually transferred divided by the time of transfer. I calculate this volume by summing over the volumes of granules transferred

since (1) all the liquid that is transferred may not form a single individual (e.g., if part of it was transferred elsewhere from c_2 during "$(t_b..t_e)$"), and (2) granules share no parts, so I will get an accurate measurement of volume. Since the number of granules transferred is discrete, I place a minimum length on the time interval over which this rate can be calculated -- this minimum being large enough so that a reasonably large number of granules are certain to have transferred. If these intervals are allowed to be arbitrarily small, inaccurate measurements can result. Let "Δt_{Lx}" denote this minimum interval length, which I set to one-tenth second.

$$[\forall r,c_1,c_2,t_b,t_e] \qquad (7)$$
$$[r=\text{liq.xfer.rate}(c_1,c_2,t_b,t_e) \leftrightarrow \text{Liq.xfer}(c_1,c_2,t_b,t_e) \wedge$$
$$t_e-t_b \geq \Delta t_{Lx} \wedge$$
$$r=\frac{1}{t_e-t_b} * \text{vol.gset}(\psi(c_1,c_2,t_b,t_e)(t_e))]$$

where

$$\psi(c_1,c_2,t_b,t_e)(t_e)= \qquad (8)$$
$$\{x|\text{Granule}(x) \wedge$$
$$[\exists t_1 \in (t_b..t_e), t_2 \in (t_b..t_e)]$$
$$[t_1<t_2 \wedge \text{Liquid}(x)_{(t_1..t_2)} \wedge$$
$$\text{Contains}(c_1,x)(t_1) \wedge \text{Contains}(c_2,x)(t_2)]\}$$

and where "vol.gset(x)(t)" is just the volume of a set of existing granules, x, at time t.

$$[\forall x,y,t][y=\text{vol.gset}(x) \leftrightarrow \qquad (9)$$
$$\text{Set}(x) \wedge [\forall z \in x][\text{Granule}(z) \wedge \text{Exists}(z)(t)] \wedge$$
$$y=\sum_{z \in x} \text{vol}(z)(t)]$$

"Set(x)" is true iff x is a set.

I define the effect of a "Faucet.flow" process to be that, if the faucet is fully open, water transfers from the faucet's supply container to a receiving container at the rate of one-quarter cup per second. If it is partially open, the rate is between one-sixtieth and one-quarter cup per second (this is idealized to simplify its presentation). The following describes the effect of a "Faucet.flow" process, p, that is occurring during "$(t_1..t_2)$" (remember, for p to occur, the faucet must not be closed). Let "faucet.of.flow(p)" denote the faucet involved with p. c, r and k are introduced to simply the formula.

$$\text{Liq.xfer}(c,r,t_1,t_2) \wedge \qquad (10)$$
$$[\text{Open.knob}(k)_{(t_1..t_2)} \rightarrow$$
$$\text{rate.liq.xfer}(c,r,t_1,t_2)=\frac{\text{cups}(1)}{\text{seconds}(4)}] \wedge$$
$$[\sim(\text{Open.knob}(k)_{(t_1..t_2)}) \rightarrow$$
$$\frac{\text{cups}(1)}{\text{seconds}(60)} \leq \text{rate.liq.xfer}(c,r,t_1,t_2) \leq \frac{\text{cups}(1)}{\text{seconds}(4)}]$$

where
"c" is "supply.cont.of.faucet(faucet.of.flow(p))(t)"
"r" is "receptacle.of.flow(p)(t)"
"k" is "knob.of.faucet(faucet.of.flow(p))(t)"

"receptacle.of.flow" is a function that is defined using geometrical primitives; I will not discuss it in this paper except to state that, for a "Faucet.flow" process, it refers either to the faucet's sink or to an open container directly below the faucet's head. For the formulas that follow, I will use "Effect(p)(t_1,t_2)" to refer to Formula 10.

The effect of a water flow process is written over an interval of time because there is a discrete quantity being measured, as I explained above. For this reason, I will place a minimum length on the intervals over which the effect of a faucet flow process is calculated (as will be seen in Formula 15). Let "Δt_{eff}" denote this minimum, which, like "Δt_{Lx}", is one-tenth second. For simple process types whose effects can be measured on a continuum, "Δt_{eff}" is zero, making it possible to describe such process types using instantaneous rates, if desired. I note that enabling conditions are expressed over intervals for similar reasons, although for the enabling condition of "Faucet.flow", there is no need for a minimum length interval.

There are 5 essential properties of simple process types. For each, I include a formula written for "Faucet.flow" that describes the property. Each simple process type will have 5 similar formulas.

1. An instance begins when (or just after) the enabling condition goes from false to true for some set of players. t_b represents the beginning time for a process.

$$[\forall f:\text{Kitchen.faucet},t_b] \qquad (11)$$
$$[\sim[\exists t][t<t_b \wedge \text{Faucet.not.closed}(f)(t,t_b)] \wedge$$
$$[\exists t][t>t_b \wedge \text{Faucet.not.closed}(f)(t_b,t)] \rightarrow$$
$$[\exists p][\text{Faucet.flow}(p) \wedge f=\text{faucet.of.flow}(p) \wedge$$
$$[\forall t][t<t_b \rightarrow \sim\text{Occurs}(p)(t)] \wedge$$
$$[\forall t][t>t_b \wedge \text{Faucet.not.closed}(f)(t_b,t) \rightarrow$$
$$\text{Occurs}(p)_{(t_b..t)}]]]$$

i.e., for appropriate t_b's, a faucet flow process begins at t_b whose player -- its faucet -- is f and which continues while the faucet is not closed.

2. An instance continues as long as the enabling condition remains true for those players.

$$[\forall f:\text{Kitchen.faucet},t_1,t_2] \qquad (12)$$
$$[t_1<t_2 \wedge \text{Faucet.not.closed}(f)(t_1,t_2) \rightarrow$$
$$[\exists p][\text{Faucet.flow}(p) \wedge f=\text{faucet.of.flow}(p) \wedge$$
$$\text{Occurs}(p)_{(t_1..t_2)}]]$$

3. An instance ends when (or just before) the condition first becomes false after the process starts for those players. t_e represents the ending time for the process.

$$[\forall f:\text{Kitchen.faucet},t_e] \qquad (13)$$
$$[[\exists t][t<t_e \wedge \text{Faucet.not.closed}(f)(t,t_e)] \wedge$$
$$\sim[\exists t][t>t_e \wedge \text{Faucet.not.closed}(f)(t_e,t)] \rightarrow$$
$$[\exists p][\text{Faucet.flow}(p) \wedge f=\text{faucet.of.flow}(p) \wedge$$
$$[\forall t][t>t_e \rightarrow \sim\text{Occurs}(p)(t)] \wedge$$
$$[\forall t][t<t_e \wedge \text{Faucet.not.closed}(f)(t,t_e) \rightarrow$$
$$\text{Occurs}(p)_{(t..t_e)}]]]$$

i.e., for appropriate t_e's, a faucet flow process ends at t_e whose player -- its faucet -- is f and which has continued for as long as the faucet has not been closed.

4. If two individual simple processes of the same type and with the same players overlap in the times of

their occurrences, they are the very same process.

$$[\forall p_1{:}\text{Faucet.flow}, p_2{:}\text{Faucet.flow}] \qquad (14)$$
$$[\text{faucet.of.flow}(p_1)=\text{faucet.of.flow}(p_2) \wedge$$
$$[\exists t][\text{Occurs}(p_1)(t) \wedge \text{Occurs}(p_2)(t)] \rightarrow p_1=p_2]$$

5. The effect applies to the players while the process occurs over intervals larger than the given minimum length.

$$[\forall p{:}\text{Faucet.flow}, t_1, t_2] \qquad (15)$$
$$[\Delta t_{eff} \leq t_2-t_1 \wedge \text{Occurs}(p)_{(t_1..t_2)} \rightarrow \text{Effect}(p)(t_1,t_2)]$$

This knowledge allows the robot to determine when faucet flow processes begin, continue and end. It provides identity criteria for these processes and it describes their effect in the real world. Thus, the robot is well equipped to plan to control such processes. In Section 5, this knowledge is used to show the effectiveness of an I/O program.

4. Robot Perception and Action

Any robot that plans must know the consequences of executing its perceptual and action routines, i.e., its own I/O programs. In this section, I specify the I/O functionality of a hypothetical robot as part of PFR.

In order to describe the effects of executing programs, a model of the robot's internal state and capabilities is needed. The robot can move about, grasp certain kinds of objects with its (single) arm and hand, and can determine certain kinds of situations by "looking" through its (single) camera eye. Let "Near(x)(t)" be true iff the robot is near object x at t. To be near an object means that the robot is able to see it and reach it. "Grasped(x)(t)" iff the robot is grasping object x at t. In order to be grasped, the object must be of a certain shape, which I denote with "Graspable(x)(t)". Only one object can be grasped at a time. In order to represent the robot's ability to identify and find objects at given times, I introduce "Identifiable(x)(t)", which partially models the robot's internal memory state.

The I/O language includes calls to primitive input and output procedures, sequencing, compound statements, if-then-else statements and while loops. Output procedure calls are program statements. Input procedure calls are program functions. There is no assignment statement. Constants denote individuals such as physical objects or instants of time. For simplicity, I assume that the execution of the control portion of statements takes zero time. This includes calls to input procedures, so they also take zero time to execute. Also for simplicity, output procedures take fixed, greater-than-zero time to execute. In the descriptions that follow, each output procedure takes 2 seconds. (For a full specification, see [Schmolze 86].) Let "E(S)(t_1,t_2)" denote the execution of statement S by the robot where execution begins at t_1 and ends at t_2, such that a new statement can begin executing at t_2.

grasp x: If x is identifiable, graspable, near the robot and nothing is already grasped, the robot will grasp x.

$$[\forall x,t_1,t_2][\text{E(grasp x)}(t_1,t_2) \rightarrow \qquad (16)$$
$$t_2-t_1=\text{seconds}(2) \wedge$$
$$(\text{Identifiable}(x)(t_1) \wedge \text{Near}(x)(t_1) \wedge$$
$$\text{Graspable}(x)(t_1) \wedge \sim[\exists y][\text{Grasped}(y)(t_1)]$$
$$\rightarrow \text{Grasped}(x)(t_2))]$$

open.knob k: If k is a faucet knob that is currently being grasped, this causes the robot to move k (if necessary) to its open position. It takes 2 seconds. For simplicity, I assume that the robot knows the current open position for k. If k is already open, the robot takes no action. If k is not open, it begins to move k immediately. At some point during execution of this procedure, k is in the open position, after which the robot stops moving it. Before describing "open.knob", I define "Stationary(x)(t_1,t_2)" to be true iff x does not change location from t_1 through t_2.

$$[\forall x,t_1,t_2][\text{Stationary}(x)(t_1,t_2) \leftrightarrow \qquad (17)$$
$$[\forall t \in (t_1..t_2)][\text{occ.space}(x)(t)=$$
$$\text{occ.space}(x)(t_1)]]$$

$$[\forall k{:}\text{Faucet.knob}, t_1, t_2] \qquad (18)$$
$$[\text{E(open.knob k)}(t_1,t_2) \rightarrow t_2-t_1=\text{seconds}(2) \wedge$$
$$(\text{Grasped}(k)(t_1) \wedge \text{Open.knob}(k)(t_1) \rightarrow$$
$$\text{Open.knob}(k)_{(t_1..t_2)} \wedge \text{Stationary}(k)(t_1,t_2)) \wedge$$
$$(\text{Grasped}(k)(t_1) \wedge \sim\text{Open.knob}(k)(t_1) \rightarrow$$
$$[\forall t \in (t_1..t_2)][\text{occ.space}(k)(t) \neq$$
$$\text{occ.space}(k)(t_1)] \wedge$$
$$[\exists t \in (t_1..t_2)][\text{Open.knob}(k)(t) \wedge$$
$$\text{Open.knob}(k)_{(t..t_2)} \wedge$$
$$\text{Stationary}(k)(t,t_2)] \wedge$$
$$[\forall t \in (t_1..t_2)][\text{Open.knob}(k)(t) \rightarrow$$
$$\text{Open.knob}(k)_{(t..t_2)}])]$$

close.knob k: If k is a faucet knob that is currently being grasped, this causes the robot to move k (if necessary) to its closed position. It is very similar to the "open.knob" procedure.

$$[\forall k{:}\text{Faucet.knob}, t_1, t_2] \qquad (19)$$
$$[\text{E(close.knob k)}(t_1,t_2) \rightarrow t_2-t_1=\text{seconds}(2) \wedge$$
$$(\text{Grasped}(k)(t_1) \wedge \text{Closed.knob}(k)(t_1) \rightarrow$$
$$\text{Closed.knob}(k)_{(t_1..t_2)} \wedge \text{Stationary}(k)(t_1,t_2)) \wedge$$
$$(\text{Grasped}(k)(t_1) \wedge \sim\text{Closed.knob}(k)(t_1) \rightarrow$$
$$[\forall t \in (t_1..t_2)][\text{occ.space}(k)(t) \neq$$
$$\text{occ.space}(k)(t_1)] \wedge$$
$$[\exists t \in (t_1..t_2)][\text{Closed.knob}(k)(t) \wedge$$
$$\text{Closed.knob}(k)_{(t..t_2)} \wedge$$
$$\text{Stationary}(k)(t,t_2)] \wedge$$
$$[\forall t \in (t_1..t_2)][\text{Closed.knob}(k)(t) \rightarrow$$
$$\text{Closed.knob}(k)_{(t..t_2)}])]$$

release: The robot releases whatever is being grasped. It takes 2 seconds.

$$[\forall t_1,t_2][\text{E(release)}(t_1,t_2) \rightarrow \qquad (20)$$
$$t_2-t_1=\text{seconds}(2) \wedge \sim[\exists y][\text{Grasped}(y)(t_2)]]$$

Less.full(C,P): An input procedure that is true iff container C is less than a certain fraction full of solid and/or liquid material; P is the fraction. If P is 1, then this is true whenever C is not full. C must be identified beforehand and the robot must be near it. The robot estimates the value of this function using its visual capabilities along with knowledge of the container's shape. However, for this paper, this ability of the robot is idealized. Let "ϕ(P)(t)" be true iff the evaluation of input procedure P at time t would be true.

$$[\forall t][\text{Identifiable}(C)(t) \wedge \text{Near}(C)(t) \rightarrow \quad (21)$$
$$\left(\circ(\text{Less.full}(C,P))(t) \leftrightarrow \frac{\text{vol.gset}(Z)(t)}{\text{contained.vol}(C)(t)} < P\right)]$$

where
"Z" is "{x|Granule(x) ∧ Contains(C,x)(t) ∧ ~Gas(x)(t)}"

Here, "contained.vol(x)(t)" denotes the maximum volume of liquid material that x can contain at time t.

5. Filling a Pot with Water

In this section, I present an I/O program that, when executed under given conditions, will cause the robot to partially fill a pot with water. The given conditions are that a pot (P) is upright, in a sink (S), and under the head of a faucet (F) that is controlled by a knob (K) with a water supply (W) that is stored in a supply container (C). K is in the closed position. The robot is near the faucet.

FP: S_1: grasp K; *(22)*
S_2: open.knob K;
S_3: while Less.full(P,0.5) do idle.for seconds(0.1);
S_4: close.knob K;
S_5: release;

When FP is executed, the robot grasps K and moves K to the open position. At this point, water begins flowing into P. In S_3, the robot waits until the accumulated water occupies more than half of P. The robot then closes K and releases it, leaving P about half full of water.

PFR can be used to show the effectiveness of the FP program. The ontology and theories presented so far will be used to show that each statement of FP, when executed, produces a set of conditions needed for the next statement execution, and that at the end, the FP program has caused the robot to partially fill P with water. Furthermore, I will demonstrate how the robot has the knowledge to infer the identity of a faucet flow process, even though no such process is mentioned in the FP program. I will only sketch a proof in this paper. (A full proof, excluding program termination, of a similar I/O program can be found in [Schmolze 86].)

I introduce T_0 through T_5, where S_1 is executed from T_0 through T_1, S_2 is executed from T_1 through T_2, etc. The relevant given conditions are:

Faucet(F) ∧ Pot(P) ∧ *(23)*
K=knob.of.faucet(F)(T_0) ∧ W=supply.of.faucet(F)(T_0) ∧
C=supply.cont.of.faucet(F)(T_0) ∧
Contains(C,W)(T_0) ∧ vol(W)(T_0)>cups(1000) ∧
Exists(F)(T_0) ∧ Exists(K)(T_0) ∧ Exists(P)(T_0) ∧
Exists(C)(T_0) ∧ Exists(W)(T_0) ∧
contained.vol(P)(T_0)=cups(1) ∧ All.water(W)(T_0) ∧
Identifiable(P)(T_0) ∧ Identifiable(K)(T_0) ∧
Near(P)(T_0) ∧ Near(K)(T_0) ∧ Graspable(K)(T_0) ∧
Closed.knob(K)(T_0) ∧ ~[∃y][Grasped(y)(T_0)]

Here I have used "Pot", which denotes a basic type for kitchen pots, and "All.water(x)(t)", which is true iff x is composed entirely of water granules at time t (definition not shown here).

The goal is that P contains at least half a cup of water at time T_G.

[∃l][Exists(l)(T_G) ∧ All.water(l)(T_G) ∧ *(24)*
Contains(P,l)(T_G) ∧ vol(l)(T_G)>cups(0.5)]

Throughout this proof sketch, I will need to make default assumptions. However, I have not investigated theories for making appropriate default assumptions in this research. Instead, I will simply make those assumptions that are needed and reasonable. As a result, I have a set of examples that a theory for making default assumptions must be able to produce. My first assumptions correspond to conditions that will not change throughout the execution of FP.

Default assumption: *(25)*
[∀t∈(T_0..T_5)]
[K=knob.of.faucet(F)(t) ∧ W=supply.of.faucet(F)(t) ∧
C=supply.cont.of.faucet(F)(t) ∧
Contains(C,W)(t) ∧ vol(W)(t)>cups(1000) ∧
Exists(F)(t) ∧ Exists(K)(t) ∧ Exists(P)(t) ∧
Exists(C)(t) ∧ Exists(W)(t) ∧
contained.vol(P)(t)=cups(1) ∧ All.water(W)(t) ∧
Identifiable(P)(t) ∧ Identifiable(K)(t) ∧
Near(P)(t) ∧ Near(K)(t) ∧ Graspable(K)(t)]

Additional assumptions are needed in a complete proof, such as that certain objects do not move throughout, that the open and closed positions for K do not change, etc.

After executing S_1, the knob K is grasped, i.e., "Grasped(K)(T_1)". This follows trivially since the given condition in Formula 23 satisfies the condition of Formula 16.

While executing S_2, the robot moves K (the currently grasped object) to its open position. Let T'_1 denote the instant that K first becomes fully open, after which it remains open. T'_1 is in the interval "(T_1..T_2)". Also, according to Formula 18, the robot begins to move K immediately at T_1.

$\text{Open.knob(K)}_{(T'_1..T_2)}$ ∧ *(26)*
[∀t∈(T_1..T'_1)][~Open.knob(K)(t) ∧ ~Closed.knob(K)(t)]

For similar reasons, during the execution of S_4, there is some instant when K becomes fully closed and remains closed (using Formula 19). Let this instant be T'_3, which is in the interval "(T_3..T_4)".

$\text{Closed.knob(K)}_{(T'_3..T_4)}$ ∧ *(27)*
[∀t∈(T_3..T'_3)][~Open.knob(K)(t) ∧ ~Closed.knob(K)(t)]

I will now show that a "Faucet.flow" process begins at T_1 and ends at T'_3. However, first I make the default assumption that K remains fully closed during "(T_0..T_1)", fully open during "(T_2..T_3)", and fully closed during "(T_4..T_5)".

Default assumption: *(28)*
$\text{Closed.knob(K)}_{(T_0..T_1)}$ ∧ $\text{Open.knob(K)}_{(T_2..T_3)}$ ∧
$\text{Closed.knob(K)}_{(T_4..T_5)}$

As a result, K is fully closed before T_1 and it is not fully closed just after T_1 (note that nothing needs to be said about K's status precisely at T_1). This satisfies the left side of Formula 11 with "t_b=T_1", leading me to conclude that there is a "Faucet.flow" process, which I'll call FF, with F as its "faucet.of.flow", that begins at T_1 and continues while K is not closed. However, Formula 11 will not let me conclude that FF ends at T'_3; Formula 13 is needed to determine process endings. Letting "t_e=T'_3" in Formula 13, I conclude that a "Faucet.flow" process, which I'll call FF2, has F as its "faucet.of.flow", ends at T'_3, and has continued for as long as K has not been closed. Of course, there is

only one process here, which is concluded from Formula 14. Since FF and FF2 use the same faucet, F, and their occurrences overlap (e.g., at T_3), then "FF2=FF".

$$\text{Faucet.flow(FF)} \wedge \text{F=faucet.of.flow(FF)} \wedge \quad (29)$$
$$\text{Occurs(FF)}_{(T_1..T'_3)} \wedge [\forall t][t<T_1 \rightarrow \sim\text{Occurs(FF)(t)}] \wedge$$
$$[\forall t][t>T'_3 \rightarrow \sim\text{Occurs(FF)(t)}]$$

Thus, the robot can identify a faucet flow process and can determine its times of occurrence.

Given the times of occurrence of FF, I can now determine its effect. First, I assume that P receives the water flowing from F (space does not allow a discussion of the necessary geometry).

$$[\forall t \in (T_1..T'_3)][\text{P=receptacle.of.flow(FF)(t)}] \quad (30)$$

By applying the formula describing the effects of "Faucet.flow", Formula 15, to the above times for FF's occurrence, 29, I conclude that a liquid transfer took place from C to P during "$(T_1..T'_3)$".

$$\text{Liq.xfer}(C,P,T_1,T'_3) \quad (31)$$

So, granules are accumulating in P that come from C (i.e., are part of what was W). From this, I can conclude that water is accumulating in P (and if I added more theories, that this water has properties similar to those of W, such as being either hot or cold). Also, given that FF is occurring, I can conclude the approximate rates of transfer. During "$(T'_1..T_2)$", it transfers at the maximum rate of 1 cup every four seconds. During the other times it transfers at a rate somewhere between 1 cup per minute and 1 cup per 4 seconds.

I now make the default assumptions that the liquid transferred by FF remains in P throughout execution of FP and that it remains liquid. Also, any non-gaseous object in P during execution of FP came from F's water supply, W.

Default assumption: (32)

$$\Big[\forall x,t \in (T_0..T_5)\Big]$$
$$\Big[\Big(\text{Liquid(x)(t)} \wedge \text{Contains(P,x)(t)} \rightarrow$$
$$[\forall t' \in (t..T_5)][\text{Liquid(x)(t')} \wedge \text{Contains(P,x)(t')}]\Big) \wedge$$
$$\Big(\sim\text{Gas(x)(t)} \wedge \text{Contains(P,x)(t)} \rightarrow$$
$$\text{gset(x)(t)} \subseteq \text{gset(W)}(T_0)\Big)\Big]$$

Given the above, I conclude that P will continue to fill with water and that, eventually, "Less.full(P,0.5)" will be false. In fact, this will happen between 0 and 2 seconds after T_2, taking into account the varying rate of water flow and the fact that the time of T'_1 is not precisely known. Therefore, S_3 takes between 0 and 2 seconds to execute, and the entire program takes between 8 and 10 seconds. So, the robot should begin execution at "$T_0=T_G-\text{seconds}(10)$" to be sure P will be filled in time. It turns out that during the execution of S_4, another half cup of water could flow, so P will be between half and completely full.

I am nearly at the given goal, Formula 24, but it is stated in terms of a liquid object and not in terms of a set of liquid granules that are contained in P. However, Formula 4 lets the robot identify the liquid in P as a physical object, and so the goal is achieved.

6. Conclusions

Physics For Robots (PFR) represents the everyday physics that a robot needs to use in planning to perform everyday tasks. Using a PFR representation scheme, a robot can reason about natural processes as well as actions. It can take into account the time events take, the gradual changes they cause and the fact that many processes, once initiated, continue without further attention. Therefore, it can plan to control many processes simultaneously. PFR also specifies identity criteria for physical objects that break apart, come together, mix, or come into or go out of existence. Therefore, the robot can plan to recognize and manipulate objects undergoing transformations, and to determine the properties of these objects based on their material composition.

The contributions of this research are:

- a strategy to develop and evaluate representations of everyday physics for robot planning,
- a general representation for part of everyday physics: including an ontology of time, space, physical objects and events, theories governing processes, material composition, etc.
- an application specific representation: describing everyday phenomena from cooking, such as water flow from a faucet, etc.

The crucial research to be done next is not only to extend these types of representations to more areas, but to use these results to design reasoning mechanisms that will allow robots to plan for everyday tasks.

7. Acknowledgements

Many, many thanks go to David Israel, David McDonald, Candy Sidner, Brad Goodman, N. S. Sridharan, Andy Haas, Marc Vilain and Krithi Ramamritham for their ideas and comments.

REFERENCES

[Forbus 85] Forbus, K. D.
The Role of Qualitative Dynamics in Naive Physics.
In *Formal Theories of the Commonsense World*, pages 185–226. Ablex, 1985.

[Hayes 85a] Hayes, P.
The Second Naive Physics Manifesto.
In *Formal Theories of the Commonsense World*, pages 1–36. Ablex, 1985.

[Hayes 85b] Hayes, P.
Naive Physics 1: Ontology for Liquids.
In *Formal Theories of the Commonsense World*, pages 71–108. Ablex, 1985.

[Hendrix 73] G.G. Hendrix.
Modeling Simultaneous Actions and Continuous Processes.
Artificial Intelligence 4:145–180, 1973.

[Schmolze 86] Schmolze, J. G.
Physics For Robots.
PhD thesis, University of Massachusetts, February, 1986.
(Also BBN Laboratories Report No. 6222, July 1986).