Affordance and Symmetry

Harold Thimbleby

University College London
h.thimbleby@ucl.ac.uk

Abstract. Whilst it is generally accepted as a positive criterion, affordance only gives the weakest of hints for interactive systems designers. This paper shows how useful it is to consider affordance as generated by a correspondence between program symmetries and user interface symmetries. Symmetries in state spaces (for instance, as might be visualised in statecharts) can be carried through to user interfaces and into user manuals, with beneficial results. Exploiting affordances, understood in this way, in addition to their well known user interface benefits, makes programs simpler and more reliable, and makes user manuals shorter.

1 Introduction

Affordance was introduced by the influential ecological psychologist J. J. Gibson [4]. Gibson introduced affordance as a provocative yet quite vague concept, and this has led to subsequent research attempting to pin down the concept adequately to exploit it further. Norman's classic *Psychology of Everyday Things* [7] brought affordance to the attention of designers. Gaver [2] further widened the scope of affordance to the design of interactive computer systems, such as graphical user interfaces.

Gibson's view is that the environment is a surface that separates individuals from physical objects. The environment is perceived, which suggests that the values and meanings of objects can themselves be directly perceived. We say that an object may *afford* some or several sorts of action, and when it does so this is in some sense a set of natural or "easy" relations. The classic example is the door plate and door handle, which when used appropriately afford pushing and pulling the door. Occasionally one comes across doors with handles that can only be opened by pushing; occasionally one comes across doors with plates that cannot be opened by pushing. The lack of affordance in each case is frustrating.

Because affordance appears to be a simple idea and represents an unequivocably "good thing" it has become a very popular design concept. Yet in 1999 Norman wrote how designers had misunderstood affordance [8]. Norman now emphasises distinctions between the user's conceptual model, physical constraints, conventions (such as cultural conventions), and the difference between perceived and real affordances. These concrete distinctions are of less relevance to the more abstract discussion of this paper, because we are not concerned here with the mechanisms (physical, cultural or whatever) that bring about a relation between the properties of the environment and the cognitive models of the user, merely

that there can be, or may not be, a relation. It is beyond the scope of this paper to speculate about the cognitive processes — beyond pointing out that the experience of "bad affordance" is so common and widely appreciated that the relevance of affordance to good design is an uncontentious issue.

Many interactive systems are notorious for being difficult to use. Poor design, without any deeper analysis, is an easy scapegoat for interaction failures that result in varying degrees of loss to users, whether using video recorders or aircraft flight management systems. Can affordance be recruited more effectively to interactive systems design? Can the concept be tightened to have a more rigorous value in design? We believe so, and this paper describes how.

2 Symmetry

We are most familiar with the concept of symmetry in the spatial and visual domains, perhaps most especially as occurring in two dimensional pictures and patterns. For example, a reflection symmetry is a feature of an object that is unchanged when it is reflected, as in a mirror. Human faces have a vertical bilateral mirror symmetry, and more symmetric faces are more attractive, possibly because a symmetric face induces less cognitive load to memory. Facial asymmetries, which are not so attractive, arise mainly through imperfections. Imperfections to one's appearance may be caused by disease or trauma, and such accidents rarely have any cause to maintain symmetries. Indeed in the natural world, threats to survival are never specialised to the left or right: when one survives, say, a left-handed threat one's chances of survival are doubled by assuming the lesson learnt should be symmetrical. To some extent, evidently, symmetry has evolutionary significance, which goes some way to explaining the widespread appeal of symmetry, including in more abstract domains such as in patterns and even rhythms. Culturally, symmetry has deep æsthetic significance, and of course is exploited in art in the widest sense, including visual arts, music and rhetoric. In the search for simplicity and power, symmetry is one of the best tools available [3].

As Hermann Weyl put it, symmetry occurs most generally when a property of an object is unchanged through a transformation of the object [17]. Of course, most transformations change objects in one way or another, but when they do not change some properties a symmetry is involved.

For example we can describe a picture of a face as a function $p(x, y)$, which tells us what colour to paint at coordinates (x, y). The face would be mirror symmetric about the line $x = 0$ if there was no change in the picture if we transformed x to $-x$. In other words, the transformation (x, y) to $(-x, y)$ leaves the property p (in this case, the face on the picture) unchanged. More specifically, the fact that $p(x, y) = p(-x, y)$ means p is symmetric in Weyl's precise sense, and of course also in the conventional mirror sense. This mathematical description of symmetry clearly captures the essence of the visual or physical symmetries of objects; it also shows how symmetry can be defined formally, just in terms of abstract transformations.

Very often we are interested in particular sorts of symmetry, and we define an S symmetry as arising when a property p of an object is unchanged through an S transformation. Mirror symmetries arise through mirror transformations, and so on. In the example above, the mirror symmetry was the S transformation (x, y) to $(-x, y)$, which is reflection about the line $x = 0$, and the property was the collection of graphical information in the picture p itself. Because of their importance, different words are used in different fields to describe the special properties involved in certain symmetries, *invariant* being a common term. We will see others below.

Physics might be called the science of natural symmetries,[1] and is concerned with many symmetries that are not visual at all. For example, special relativity arises when constant velocity transformations leave the property the speed of light unchanged.[2] Chemistry is the study of properties that are unchanged when objects (which chemists call atoms) are replaced with other objects of the same class (which chemists call elements). Chemistry makes a good example of how agreement on "better" symmetries advances reasoning: the conceptual move from heat, fire, water and air — through phlogiston and other conceptual models — enabled human reasoning to be more effective, reliable and indeed easier. Simply, in this paper, we want to use symmetry to help design, understand and use interactive systems, rather than chemical processes.

Mathematics has very general ideas of symmetry, expressed for instance in group theory. Felix Klein famously defined geometry in 1872 as the study of properties of figures that remain invariant under particular groups of transformation. In particular Euclid's geometry is the study of the so-called rigid transformations, such as rotation and reflection, that preserve area and distance. Interestingly, Euclid himself did not specify the essential symmetry axiom that geometrical properties are unchanged when a figure is moved about in space.

Symmetry represents key concepts. The laws of physics are invariant under translation (change of position): so we expect the laws of physics to be found unchanged if we did our experiments a million miles away. Yet the laws of physics are very different just a metre below where I am writing: few of the experiments I can perform on my desk work inside the concrete floor — for obvious reasons! Symmetries can be restored by translating more physical properties of my desk to the floor.

To a great extent, then, science advances by studying the ways in which symmetries are broken, and then finding newer more general symmetries to restore the elegance. Mathematics often advances by developing notations that are invariant under transformations; for example, vector algebra is a powerful tool (e.g., for physics) precisely because it is insensitive to rotation and translation of coordinate systems. Furthermore, because it is invariant vector notations

[1] Noether's Theorem shows a deep connection between symmetry, least action and conservation laws, and is one of the most profound theoretical results behind modern physics.

[2] Newton's laws of motions are invariant too, but they predict the speed of light varies with the observer's motion.

need not, and in fact generally do not, mention coordinates, and therefore become easier to use by removing irrelevant detail. In computing, symmetry arises in many areas, such as mobile computing (computing facilities are unchanged through transformation of place) and in declarative programming (which we discuss below).

Symmetry sounds ubiquitous, and if it was it would be a pretty vacuous concept. We are generally interested in things that change and how they change, and therefore by definition we are interested in broken symmetries, more than in the symmetries themselves, except in so far as they provide a "background" that highlights the changes. Certainly some symmetries seem trivial (for example, we are so familiar that movement in space leaves things unchanged that translation symmetry seems obvious). Other symmetries, though, seem more profound. Consider scale symmetry, where things are unchanged through a transformation in size. In the physical world, scale symmetry does not apply. Things are particular sizes, and they do not work or do not work well at other scales (though there are approximate scale invariants in fractals). Atoms are necessarily a certain size, and things therefore cannot be made too small; when things are made too large, their strength — which does not increase at the same rate as their mass — becomes insufficient to support them. Animals the size of insects can fly; we can't because we operate at a larger scale. And so on.

User interfaces for many applications, however, would be improved if they were scale symmetric (be the same even when you change their size), even though strictly speaking there is no corresponding physical symmetry to apply. People with limited visual acuity could "zoom in" without affecting any other aspect of the user interface. Images could be scaled so they could be read easily at any distance, or projected for an audience scaled to screens of any size. Unfortunately many auditorium projectors use digital display technology (e.g., LCD screens) and scaled images cause aliasing problems, often with the result that text becomes unreadable when projected at "the wrong resolution."

3 Affordance and Symmetry

Gibson wished to explain vision by positing some higher order features of vision that are invariant with motion and rotation and are "picked up" by the observer. He held the view that the function of the brain was to "detect invariants" despite changes in "sensations." With this conception he freed vision research of what computer scientists would call implementation bias — an idea later formalised by David Marr in his three-level model of visual processing into computational, representation/algorithm, and hardware implementation [6]. As Marr points out, whereas Gibson thought of the brain as "resonating" to the invariants, the detection of physical invariants is in fact an information processing task.

A typical wood pencil has two main symmetries. It has a hexagonal or sometimes circular symmetry along its long axis, and it has an approximate reflectional symmetry about its centre (strictly, about a plane through the centre and

orthogonal to the major axis). Because of the rotational symmetry, it does not matter what angle a pencil is grabbed at. Because of the approximate reflectional symmetry, it is easy to make errors grabbing a pencil: it may be grabbed "up side down" and be unsuitable for writing — unless it really is symmetric, that is, sharpened at both ends. Clearly the symmetries provide freedoms in the way an object can be used, and any approximate symmetries provide opportunities for errors because users tend to expect the full symmetries to work.

There are many other pencil symmetries. For example, the property "pencilness" is unchanged when we swap one pencil for another made by a different manufacturer. The colour of the wood of a pencil is another symmetry, whereas changing the colour of the lead is (for most tasks) not a symmetry. We are disappointed in pencils made by cheap manufacturers because they persuade us that the pencil has symmetries that they fail to uphold. Conversely, we are pleased by pencils whose quality of symmetries exceeds our expectations: thus one that was indefinitely unchanged as one wrote with it would be very nice, as it would have an inexhaustible lead. More plausibly, a propelling pencil is "better" because it size and shape remains unchanged as it is transformed by use — it has an additional symmetry over conventional pencils, which get shorter as they are used. A pencil that was exceedingly smooth is merely one that feels invariant on any surface. And so on.

We could define a pencil (and everything equivalent to a pencil) by its full set of symmetries, namely the set of transformations that keep all "pencilness" properties invariant. *Affordance can then be defined as those symmetries that apply under the actions relevant to the activities or tasks that are performed with the object.* Rotating a pencil about its long axis makes no difference to how well it writes, and therefore an affordance of a pencil for writing is its rotational symmetry. However, if we had the unusual task to read the writing on the pencil shaft that says who manufactured the pencil or what its hardness was (H, HB, etc) then rotation does change the task, and it is therefore not an affordance for this task. In particular, we note that the visual properties of a pencil do not afford perceiving its hardness — suggesting (if this was a concern for some task) that a new perceptual representation of hardness (e.g., colour coded bands) that was invariant under a pencil's symmetries could be a practical benefit.

In summary, some pencil symmetries are interesting, others not; not all symmetries lead to useful affordances; enforcing some affordances leads to new designs, such as the propelling pencil. Clearly affordance raises design trade-offs: what is the cost of implementing a suitable symmetry against the benefit to the user, and how should this trade-off be accounted for (what are the manufacturing set-up costs; how many users are there; what are the risks of error; and so forth)?

Pencils are a very simple example, but for cultural and economic reasons do not pose interesting design problems for us here. (See Petroski [9] for a thorough discussion of pencil design.)

4 State Spaces

All digital computer systems implement state spaces, and with the exception of continuous systems (e.g., control systems), computer systems can be defined as state space automata. Even simple computer systems have enormous state spaces, so various means are used for describing the computer system without specifying the state space explicitly as such. Programming languages provide structures (most notably data types and procedure calls) that heavily disguise the underlying state machine, so that the programmer can concentrate at any moment on very simple components of the state machine (such as a single conditional). Provided we use reliable compilers, it is not necessary to distinguish between program source code and the compiled program itself (which is what actually creates the user interface and presents the interactive behaviour to the user). We won't emphasise this distinction in what follows.

Many states are closely related, and are considered simple variations of each other. For example, the state of a ticket vending machine strictly is dependent on the length of ticket roll available for printing, but for most purposes the length of tape is immaterial. One might wish to program a ticket machine abstracting away from the symmetries. Indeed, one imagines that many ticket machines have programs in them that do not explicitly mention remaining ticket roll length, and they are therefore much simpler and more likely to be correct — except in the circumstance that the ticket machine has actually run out of paper!

There are, then, large classes of state where if the computer system is transformed from one state to another within the class, its behaviour is practically unchanged. Thus the behaviour of a ticket machine is unchanged as the ticket roll is transformed into a shorter (but still non-empty) roll. Such symmetries are conventionally represented by abstractions in the program specifying the behaviour. For example, the program may have a function that prints tickets; this function will abstract away from many concrete details of printing tickets, and will be "the same" function regardless of the length of paper in the ticket roll. The program code that represents the function is much clearer, and therefore programmed much more reliably, because it does not mention any details whose changes should not affect the meaning of the function (everything else, all the other transformations, the machine is doing).

Conventional programs often go wrong when some of these unmentioned details somehow affect the meaning of an abstraction: in some sense the symmetry is betrayed. Functional programming, further, sets out to *guarantee* a precise correspondence between the abstractions and the symmetries. As there are no hidden state variables in functional programs, a function application has the same result everywhere, dependent only on its actual parameters. (Indeed, functional programming is promoted because, for many applications, it has a better affordance, in our sense, for the task of programming than imperative programming.)

Functional programming, and declarative programming more generally, avoids referential opacity: what expressions mean should not depend on how they are referred to. In other words, a transformation of context in a functional program

leaves meaning unchanged: referential transparency is therefore a symmetry. In particular, referential transparency means parts of programs can be designed without regard for how they will be used, since their meaning is independent of their context of use. In imperative programming, in contrast, one often has to worry about concepts such as initialisation and global variables before some piece of code can be used — this makes imperative programming harder and less reliable. On the other hand, it is possible to program imperatively with any "degree" of referential transparency, and therefore the disciplined programmer can choose how to exploit the dependencies and so make the programming task easier in *other* ways.

These ideas of exploiting symmetry in programming are pursued thoroughly in [1], which, however, is concerned more with graphs and proof (i.e., logic programming). One point made is that programming is so flexible that determining simple patterns of symmetry is hard if not non-computable. Affordance, as understood in this paper, is concerned with the relation between program and user interface symmetries, rather than simply the symmetries within a program: therefore affordance is a much easier concept to apply than program symmetry without constraint.

Object oriented programming is a weakening of functional programming and has gained prominence for two main reasons:

First, there are objects in the real world (such as people) and objects can be created in programs whose behaviour, for the purposes of the program, are the same as the behaviour of the physical objects. In other words, as one transforms from the real world to the simulated world, the properties of concern (e.g., the person's salary) are unchanged. The wide range of such symmetries, and the notational support for preserving them, makes object oriented programming convenient for many sorts of programming task.

Secondly, graphical user interfaces draw images of things (also called objects) like windows and icons on computer screens and allow the user to interact with them. Object oriented programming provides a convenient notation where the visible screen objects are also objects within the program world. As one considers the transformation from the perspective of the user (figures on a screen, arrangements of pixels, etc) to the perspective of the programmer, many properties are unchanged — and thus symmetries are involved. Both the user and the programmer may agree on the "position" of a window, and they would agree that similar transformations of the position have the same effect. Furthermore the object oriented programming language provides a notation where the position and transformations of the position can be described without reference to all sorts of detail, such as how to redraw pixels, or how the size of the object affects (in immaterial ways) the positions of parts of the object.

In fact graphical user interfaces "go wrong" when the user thinks there are properties in objects but which the program has not implemented. The user assumes there are symmetries but which, unfortunately, the programmer has failed to provide. Often, an important symmetry the user believes in is that a program's behaviour is unchanged over time. If a window behaves like *this*

now, it will — or should -behave like *this* later. A badly implemented object oriented system may have memory leaks or other problems, which are almost certainly never mentioned explicitly in the program because there were *supposed* to be symmetries. The consequence may be that a window can only be moved or resized so many times before the time symmetry fails.

Serious consideration of symmetries in the user interface may well improve programming standards: an example discussed in [14] is a mobile phone. The user might expect a symmetry in the user interface: specifically, the menu navigation system should be unchanged despite transforming the user's position in the menu hierarchy. The mobile phone examined in [14] makes various changes as the menu is navigated: evidently, it was implemented in an *ad hoc* fasion (there is no plausible rationale to explain deliberate variation!), and one has less assurance of its correctness than if user interface symmetries had been exploited by better programming. Furthermore, if this mobile phone's user manual accurately reflected the variations in the user interface, which it doesn't, it would have to be longer. *If affordance is the correspondence between state space and user interface symmetries, then the correspondence can be extended to user manuals and therefore be exploited to make the manuals shorter.*

5 Example: Pushbutton Devices

Interactive pushbutton devices are ubiquitous and one can argue that they are finite state machines [16]. There may be a huge number of states in their implementation (an astronomical number if the device is connected to the internet) but the user's explicit model cannot be so large.[3] The user must rely on symmetries, that certain transformations leave the user's model unchanged. To take a simple example, a digital clock works the same way whatever time it is, though obviously each individual time it can display must represent a different state. The user may rely for some purposes on the distinctions between the various states, but the way in which the clock can be used is *essentially* unchanged as states are transformed into other states in the same class through, in this case, the passage of time. Again, the way a video recorder is used depends only very weakly on the position of the video tape: having successfully done a fast forward so the tape is about midway, the transformation represented by [play] [stop] changes nothing in the way the user interface works that a user will notice.

The physical interface (whether knobs and switches in a strict physical sense, or their visual representation on a screen, as in an aircraft glass cockpit) may present symmetries to the user that the user will assume are implemented by the system.

Let us now consider a concrete example.

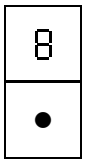
[3] The user has an implicit model, e.g., at the level of neuronal activity that involves a stochastic state machine with a vast number of states. For practical purposes this implementation model is inaccessible.

5.1 Affordance and Symmetry in a Digital Alarm Clock

A digital alarm clock has a large enough and complex enough state space to make its implementation a not completely trivial exercise. Moreover the user interfaces of real digital clocks come in a bewildering variety, and many are surprisingly difficult to use — so this is not altogether an academic example.

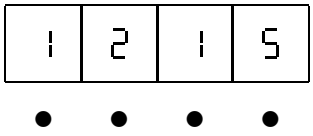
To make our discussion easier to handle, we will skip many interesting issues. For example, the clock should work the same regardless of what colour it is (a symmetry), so we do not need to worry about colour — except that we know that some users are colour blind and that for them colour choices can be crucial.

First consider a single digit display positioned above a press button. We will call this assembly a domino, here shown displaying the digit 8 (imagine it is warmly glowing in red so it can be read in the dark — perhaps the buttons glow too, so they can be found and pressed easily in the dark):



Pressing the button increases the digit by one, taking the display of 0 to 1, or from 1 to 2 . . . and so on, and from 9 back to 0. The affordance is expressed in the correspondence between the physical symmetry (a vertical movement translates the position of the button to the position of the display) and the internal state system symmetry which is *also* cyclic.

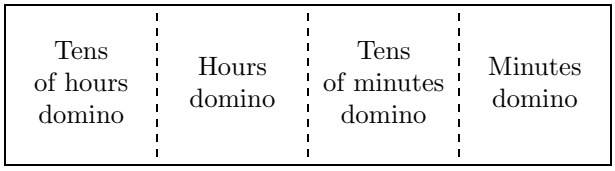
Now take one domino and copy it right three times to get a row of four dominos. This makes a simple horizontal repeating pattern. In the picture below we have not required the specific state of each domino to be copied, and thus any particular time can be displayed. Of course the set of dominos would be easier to use (if not rather trivial) if they always showed the same number, but we have to make trade-offs if we want to deal with the complexity of the real world — in this case, that time is represented by four digit numbers not always all the same. It is a bit harder to use, but far more useful!



With this four-fold repetition, we can display times easily, for instance using a conventional 24 hour notation. But as the vertical translational symmetry is preserved, buttons still correspond with the digits immediately above them. The affordance of a single domino is retained, and thanks to using the same layout there is symmetry and the "ease of use" property of one domino transfers to any other domino (ignoring the user's cognitive resource limitations: if we had enough

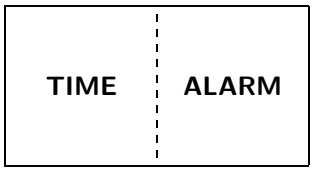
dominos, the usability would certainly decrease). We've got a user interface that controls a system that is four times more complex, yet is more-or-less as easy to use.

The statechart [5] representation of the state space, which is drawn below, has a corresponding symmetry. Each name in the statechart is the name of a process that implements the corresponding domino; the dashed vertical line is the statechart notation for allowing the processes to run in parallel. (The dominos can be represented in a statechart explicitly as simple transition diagrams, and they would then reveal their internal cyclic symmetry.)



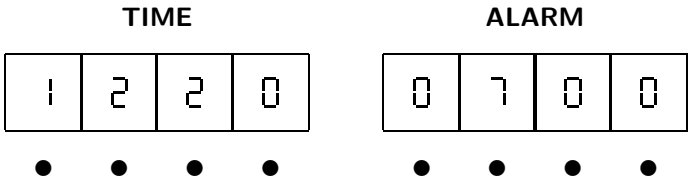
An alarm clock not only displays the time but also has an alarm setting, which is also a time. The alarm setting mechanism is a reflection of the alarm time: the state spaces of the alarm and clock are identical except that the clock has an external "tick." Even the "*time* = *alarm*" detection is symmetric, and either or both alarm and time mechanisms could ring when the alarm goes off.

If we draw the obvious statechart for the complete alarm clock we would see translational symmetry in the state space (for clarity in the next statechart, below, we have not drawn the subsidiary statecharts for each component of the alarm clock, as they are just copies of the four-component statechart above):



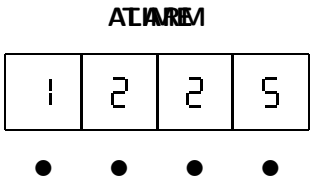
In this statechart, there are two clusters of parallel states, meaning that actions can change the states either side of the dashed line independently. That's what we mean: on the time side, tick actions (caused internally by the clock) and user actions (button presses) change state, and on the alarm side, actions (again, button presses caused by the user) change the time the alarm is set to. Either side can be changed independently by the user, but the time side will continue ticking on its own.

Given the structure of the state space, then, our understanding of affordance therefore suggests the following physical layout for the alarm clock user interface:

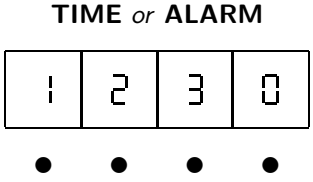


Here, by moving the alarm dominos left, they can be brought into coincidence with the time dominos: so there is a translation symmetry, just as there is in the statechart representation. As before, the physical symmetry is not just in how it looks, but is the same symmetry in how it is used. The time and the alarm buttons *all* look and work the same way. We've got a clock with a vertical symmetry (within dominos), and two sorts of horizontal symmetry (within and between blocks of four dominos).

If the classes of state are as closely symmetric as we say, why not make their representation in the user interface identical? If a manufacturer can halve the size of the gadget, and hence halve the number of buttons and digit displays yet retain functionality, there's going to be a powerful financial incentive to reduce the size of the interface. Yes, we can "fold" the user interface to take advantage of symmetry to make the alarm clock smaller:

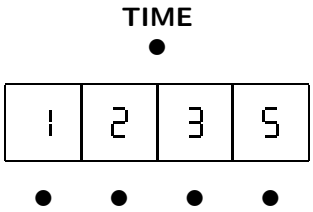


The messy overprinting of the **TIME** and **ALARM** words here shows that we have a broken symmetry. We don't know whether the display of 1225 is referring to the time or the alarm setting. (I think it's the time, as it was drawn five minutes after the last picture!) The appearance of broken symmetry is easy to fix: we put the words next to each other or erase them. We will choose to put them next to each other:



This design is obviously ambiguous, so perhaps we should have an action changing the mode of the alarm clock between showing its time or its alarm

settings. (When the clock is in alarm mode, the buttons mean "change the alarm setting" and when the clock is in time mode, the buttons mean "change the time setting.") A single button can do this. To spell it out, in alarm mode, this button means get into time mode, and in time mode it means get into alarm mode. We might think of a layout like the following, which retains the vertical symmetry between a display and its controlling button:

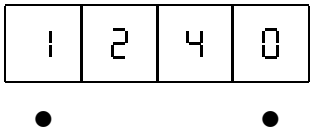


An alternative approach would be to replace the "or" with a physical knob or other control (such as a slider or rocker) to resolve the ambiguity. Here we try a small knob, and where the brightness of (in this case) ALARM has been reduced to confirm the knob is pointing away from it:

By our definition of affordance, a knob that is intended to be rotated must have rotational symmetry, which is the case here. In general, of course, knobs need not have *visible* rotational symmetry, as is the case in the picture above, although if real knobs were to work mechanically by rotation there must be an internal rotational symmetry (else they would simply not rotate). Users learn when they come across various examples that "knobness" is preserved by rotation even when a knob may have no surface properties of rotational symmetry. One issue in user interface design is providing the contextual cues that, as the case may be, "knobness" applies particularly where no visible affordances are used. (In GUI user interfaces, knobness is problematic since mice, touch sensitive screens or equivalent introduce their own ambiguities when used to rotate controls — rotating a mouse, or rotating a finger pressing on a touch sensitive screen does not transfer rotation to the control.)

Most proprietary digital clocks throw affordance to the wind, and save manufacturing costs by having a minimum number of buttons. If we also abandoned affordance to make the correspondence of the button to the action more complex — so pressing buttons can do more complex things — then we can easily get down to two buttons, and discard the **TIME** and **ALARM** mode indicators:[4]

[4] It is theoretically possible to get down to just one button and retain all the required features, but we hope this reduction would make the alarm clock so obviously hard to use even very bad designers would not be so tempted.



Of course there is a superficial physical symmetry (a mirror symmetry) but it relates to nothing about how to use the gadget, to nothing in the state space. *It is not an affordance.* Probably the buttons will be labeled, but there are so many ways to give them meanings that designers have almost got a free rein. As they can do what they like they do; and then we don't know and can't work out how it works. I guess in this case, it is likely that pressing the right-hand button increases the minute pair of digits, and pressing it for two seconds increases the hour pair of digits, and the button on the left does the same sort of thing for setting the alarm. Pressing both at once sets the alarm or unsets it. And perhaps we'll have a flashing colon in the middle of the four digits so we can tell if the clock is running or whether we are setting the alarm ... That's just one way of doing it! There are so many other ways of designing an alarm clock badly, without regard for affordance, that the user is at a loss to know how to use it. Learning one bad design is of little help for using the next, and users cannot transfer their skills.

5.2 Affordance and Symmetry in an Analogue Alarm Clock

The two digital clock digits on the left must not show a number exceeding 23, and the two on the right must not exceed 59, these being standard constraints on our notation for time. Life would be a lot easier (both for us as designers and for users) if there were 100 hours in a day, and 100 minutes in an hour: this would give us a much tighter symmetry for a digital clock! Unfortunately, our design brief is not to make time-telling *in general* easier to use, but to make clocks easier to use given long-standing cultural conventions that are immutable — we can't turn the clock back. These sorts of design constraints and tradeoffs are routine in any design process: should the designer improve the entire system and its basic assumptions, or should the designer provide the best possible user interface given that some parts of the system and some assumptions, attitudes and values are never going to be changed?

One might continue using symmetry to explore the best way of handling the different modulo arithmetic that is required for each domino. One possibility is to remove both "tens" buttons because the physical presence of these buttons suggests a state space symmetry that is not present, but the "units" buttons do work symmetrically so could be retained.

Alternatively, once might emphasise the cyclic state space symmetry: any setting plus 24 hours is the same time; and any setting plus 60 minutes is the same minute time. This state space symmetry can be carried into a physical rotational symmetry. Because there are several cyclic state space symmetries, the digital alarm clock should have several rotational affordances. Now, the only

way several rotation symmetries can be maintained together is if their centres coincide. If further we require that an increase in time is in the same direction (clockwise, to follow convention) as an increase in alarm time, the rotations must be in the same plane. It is but a short step to designing a conventional analogue style clock face. The adjustment knobs could be concentric, perhaps with the alarm setting on the front and the time on the back.

Cyclic symmetry, then, gives us the conventional affordance for clock-based interfaces and avoids many of the potential confusions with a push button interface. A clock with such cyclic symmetry *cannot* be set to an invalid time (such as 26:78 hours) and this is manifestly obvious from its display behaviour. By exploiting affordance we eliminate possible user errors.

6 General Design Heuristics as Symmetries

Many existing design heuristics can be recast as symmetries. Here we give some brief examples; further details of the heuristics can be found in the references.

Modelessness means that the way a system is used does not change over time whatever we do; if I press a button, the system still behaves the same way. So modelessness is a time-translation symmetry: my machine will be the same in the future as it is now.

Almost every interactive system distinguishes between what humans can do and what computers can do. Hand held calculators make a good example [13]: although arithmetic itself makes no distinction, calculators want humans to press buttons to create a problem, then to press = before they give an answer that is only a number. The number a calculator displays is only correct immediately after = is pressed; other transformations (such as pressing +1) change the display so that it no longer shows a correct answer. There is a radical distinction, then, between the input, key pressing, and the display of the answer. Yet, with a different design, there could obviously be a symmetry. Why not ensure the display is correct at all times, so any input (transformation of the user's sum, even if partially completed) leaves the correctness of the display unchanged? How this is possible is fully shown in [12]; the method used brings referential transparency to user interfaces — which is a key symmetry of arithmetic and therefore a very appropriate affordance for calculators.

Equal opportunity [10, 11] is the symmetry that input and output to a system are equivalent, and that they have equal status. Since most little gadgets have a few buttons and a very different sort of display (if at all), equal opportunity is a potential symmetry that is hard for the designer to apply, but in computer systems such as word processors it suggests many creative design ideas, including WYSIWYG as a special case.

It is simpler to build systems without equal opportunity, because there is no need to implement the other half of the relation. When equal opportunity is implied in the task domain but is not provided in the user interface, the results can be disastrous.

Systems typically, but restrictively, require the user to behave in particular ways, whereas the user does not distinguish between the different ways. For example, a ticket machine may require the user to choose the destination A before the discount B. To the user the transformation AB to BA leaves the world unchanged, and therefore an impermissive system that requires A first is hard to use for those users who try to do B first. The ticket machine may be designed to force A first, in the hope that the user will complete their task before applying any discount (which would lose the company money). *Permissiveness* [15] generalises these ideas, and also applies them to other non-temporal symmetries.

7 Passwords

Affordance, appropriate correspondences in symmetry makes user interfaces better. Can symmetry make user interfaces harder?

A secure system uses passwords. One password can be transformed into another and the behaviour of the system is almost always unchanged, namely it won't allow the user in unless the password happens to be exactly right. From the designer's point of view, it is very important to create a symmetry with exactly one exception, otherwise the user interface might provide clues (e.g., in the time it takes to process passwords) to a hacker. From the user's point of view there is no such symmetry because entering each password takes (say) ten seconds — and most secure systems significantly increase the time after a few attempts — and no user is prepared to consider transforming one password into another for very long.

For the authorised user their correct password is as good as any other they might have chosen — except in those infuriating systems that do not allow the user to use the password of their choice, because the system makes (no doubt well advised!) distinctions between valid and invalid passwords.

8 Affordance or Partial Affordance?

The opening example of this paper, of a door handle illustrates conventional affordance well, but it is not immediately apparent how, or even whether, this application of affordance relates to symmetry. We have left addressing this issue to last. The bulk of this paper showed that symmetry and some (possibly) restricted meaning of affordance are closely related. This still leaves open many questions. In particular: Aren't the broader ideas in affordance excluded by the symmetry formalism?

A physical object that has affordances has surface features that create (in some sense: physical, perceptual, cultural ...) the affordance. A simple push-button that affords being pressed by a finger tip probably has physical features, such as a dimple or dent in its surface (or it has a surface texture or image that suggests it is made out of a material that dents easily, or it is an on-screen picture of a button that creates the same or a similar retinal image as one that really does have the affordance). The surface of the dent matches or otherwise

conforms to the surface of a finger tip. Metaphorically, if not quite exactly, the button and the finger reflect each other.

It seems in this way, then, that any affordance can be described as a correspondence with a symmetry. But this is a sort of "hand waving" approach to affordance and symmetry. That's a serious criticism, and one we must deal with.

We are very familiar with physical and mechanical devices, such as pushbuttons and door knobs, so it is quite hard to think of them being really badly designed. Everyone, even the very worst designers, have a good idea of the design requirements and trade-offs: every pushbutton is adequate, or it would not even be recognised as a pushbutton at all — it would be a block or a bump or a mere visual pattern. Egregiously designed pushbuttons would not sell, nobody would want them as pushbuttons. At best, a pushbutton without pushbutton affordance is a broken pusbutton.

Clearly affordance circumscribes an important property of design that pushbuttons (doorknobs, letterboxes and so forth) exemplify, but affordance does not constrain the design of pusbuttons when "common sense" does even better. Put the other way around: the conceptual constraints of symmetry as an interpretation of affordance appear almost completely trivial, and hardly a clarification, when applied to pushbuttons.

The main example of this paper was of the design of a digital clock. Unlike in the design of pushbuttons there are clearly very many ways of getting a digital clock's user interface wrong or inadequate but still more-or-less "clock like." Now symmetry appears, in this more complex design context, as a useful and powerful design constraint. We conclude that symmetry being a weak description of affordance in familiar contexts is not a problem.

It's rather like someone inventing numbers [symmetry] and saying counting is helpful with dealing with apples [design]. Some people might however say they can deal with apples perfectly well already without explicit numbers, because with familiar handfulls of apples explicit counting is unnecessary. This is certainly true of the apple tree in my garden, where "none," "one" and "amazing" are sufficient concepts to cover all eventualities. If we want to run an orchard growing apples, being able to count and understand numbers more generally (addition, percentage markup ...) would be essential to stay in business. Likewise with affordance: if we do trivial design, there is no pressing need to go deeper into the concepts. If we are designing complex devices, however, we need to understand affordance in a way that we can generalise to successfully constrain the design choices open to us. We need concepts that tend to make complex products easier to use.

Just as with counting apples where having "more apples" in itself is not sufficient to mean "more profitable," being "more symmetric" in itself is not sufficient to ensure "easier to use" without further design work, which will no doubt include careful evaluation to confirm whether and to what extent the design process is achieving its goals. But being clearer on the symmetries — just like counting apples — will give us good insights into good design.

9 Conclusions

Conventionally understood, affordance is an appealing design concept but suffers from vagueness, latterly recognised in the debate in the literature. Its combined vagueness and natural appeal has ensured its widespread recognition in the design community.

Symmetry is a familiar concept, with natural appeal, but unlike affordance is readily expressed formally. This paper argued that symmetry provides a very natural way of defining affordance as linking user interface symmetry to state space (implementation) symmetry. Moreover, once this abstract symmetry is recognised, it can be applied potentially in many other areas of interaction — being abstract it is not located in a physical place: relating to the user's conceptual model, to the user documentation, to interactive help, and so on. Of course, symmetries may be made evident in the visual representation of the user interface, or mathematically in the formal specification of the interface (e.g., as represented in statecharts), or in diagrams explaining the interface to users.

Preserving symmetries in the representation of an interactive system generates affordances. Norman [8] would regard real and perceived affordances as distinct; we note, however, that symmetry can apply to visual symmetry or to mechanical symmetry. It is important to emphasise that symmetries *generate* affordances, and thus that there may be affordances other than those generated by obvious symmetries.

Thus this paper proposed a new rigorous basis for affordance, and one that can be used constructively in a precise way in the design process, and obtains various advantages, such as better user manuals and reduced user error. Specifically:

- Something is symmetrical if you can do something to it so it is in some way the same afterwards. Moving something, rotating something or turning it over but leaving it looking the same lead to translational, rotational and mirror symmetries respectively.
- Affordance is a correspondence between symmetries in the user interface and in the state space of the system. If two buttons look the same, the buttons have (visual) translational symmetry; if the buttons control state machines that are the same, the state space has a translational symmetry; if both symmetries apply, then the symmetry in the user interface affords the corresponding symmetry of the state space. The symmetries may be stronger, applying to a row of buttons, perhaps, or applying to many classes of state, and the affordance will be stronger accordingly, provided the correspondence is maintained.
- Once an affordance exists, other structural correspondences follow that may be exploited, for instance in the user manuals and in the program implementing the system.
- The user activity and task allow a designer to select, out of all possible affordances, relevant affordances for a design to represent with appropriate state space and physical symmetries. Choosing a design that represents the

relevant abstract state space symmetries in physical symmetries (whether mechanical — in conventional knobs and switches, or visual figures) therefore creates a user interface that affords the user to infer the task-relevant actions. Affordance constrains the design options and makes both design and use easier.

Once such affordances are determined, in addition to the conventional benefits of affordance in terms of usability, both program and user manual can be simplified and made more reliable. In practice, a key contribution of this new understanding of affordance will be the reduction in user interface implementation bugs; perhaps, too, because affordance and symmetry is about fundamental interaction between users, user interfaces and implementations (a point Gaver *op. cit.* [2]) also alludes to), user interface designers and programmers will be able to work more constructively together. This paper alluded to the deeper symmetries available in programs which are explicitly exploited, for instance, in object oriented programming. A major area for future work will be to develop programming styles or paradigms that increase the correspondences between symmetries visible in source code and in user interface symmetries.

Acknowledgements

Ann Blandford, Paul Cairns, Matt Jones, Jef Raskin and Ian Witten made valuable comments on the ideas expressed here. Harold Thimbleby is a Royal Society-Wolfson Research Merit Award holder, and acknowledges that generous support.

References

1. Carbone, A., Semmes, S.: *A Graphic Apology for Symmetry and Implicitness*, Oxford: Oxford Science Publications, 2000.
2. Gaver, W.: "Technology Affordances," ACM CHI'91 Conference, 79–84, 1991.
3. Gelernter, D.: *The Aesthetics of Computing*, Phoenix, 1998.
4. Gibson, J. J.: *The Ecological Approach to Visual Perception*, Boston: Houghton Mifflin, 1979.
5. Harel, D., Politi, M.: *Modeling Reactive Systems with Statecharts: The Statemate Approach*, McGraw-Hill, 1988.
6. Marr, D.: *Vision*, New York: W. H. Freeman & Company, 1982.
7. Norman, D. A.: *The Psychology of Everyday Things*, New York: Basic Books, 1988.
8. Norman, D. A.: "Affordance, Conventions, and Design," *ACM Interactions*, **VI**(3):38–43, 1999.
9. Petroski, H.: *The Pencil: A History of Design and Circumstance*, New York: Alfred A. Knopf, 1990.
10. Runciman, C., Thimbleby, H.: "Equal Opportunity Interactive Systems," *International Journal of Man-Machine Studies*, **25**(4):439–451, 1986.
11. Thimbleby, H.: *User Interface Design*, Addison-Wesley, 1990,
12. Thimbleby, H.: "A New Calculator and Why it is Necessary," *Computer Journal*, **38**(6):418–433, 1996.

13. Thimbleby, H.: "Calculators are Needlessly Bad," *International Journal of HumanComputer Studies*, **52**(6):1031–1069, 2000.
14. Thimbleby, H.: "Analysis and Simulation of User Interfaces," Human Computer Interaction 2000, BCS Conference on Human-Computer Interaction, edited by McDonald, S, Waern, Y. & Cockton, G., XIV:221–237, 2000.
15. Thimbleby, H.: "Permissive User Interfaces," *International Journal of HumanComputer Studies*, **54**(3):333–350, 2001.
16. Thimbleby, H., Cairns, P., Jones, M.: "Usability Analysis with Markov Models," *ACM Transactions on Computer Human Interaction*, **8**(2):99–132, 2001.
17. Weyl, H.: *Symmetry*, Princeton University Press, 1952.