Interoperability and objects

Michael Leyton

DIMACS Center for Discrete Mathematics & Theoretical
Computer Science,
Busch Campus, Rutgers University,
New Brunswick, NJ 08904, USA
E-mail: mleyton@dimacs.rutgers.edu

Abstract: The book *A Generative Theory of Shape* (Michael Leyton, Springer-Verlag, 2001) develops new foundations for geometry specifically designed to give a single mathematical language for the enormous range of information objects involved in the product lifecycle. This is done by giving a mathematical theory of the structure of *intelligently-generated objects*. It is argued that intelligence is characterised by maximising *transfer* and *recoverability* – which are, in fact, the basic properties of interoperability. Thus the book gives a mathematical theory of intelligence, and, in doing so, brings interoperability into the very foundations of geometry.

Keywords: PLM; interoperability; object-oriented; inheritance; re-usability; group theory; kinematics; memory storage.

Reference to this paper should be made as follows: Leyton, M. (2006) 'Interoperability and objects', *Int. J. Product Lifecycle Management*, Vol. 1, No. 2, pp.98–128.

Biographical notes: Michael Leyton is a Professor in the Center for Discrete Mathematics and Theoretical Computer Science (DIMACS) at Rutgers University. The central purpose of his work is the elaboration of a unified geometric language for the computational disciplines and physical sciences, arguing that, in the modern age, computational issues require foundations to geometry that are fundamentally different from the standard foundations based on Klein's invariants program. His new foundations for geometry are elaborated in his books, *Symmetry, Causality, Mind* (MIT Press, 630 pages) and *A Generative Theory of Shape* (Springer-Verlag, 550 pages). His mathematical theory of shape has been applied by scientists in over 40 disciplines, from chemical engineering to meteorology. He is President of the International Society for Mathematical and Computational Aesthetics.

1   Introduction

The integration of heterogeneous software systems within a large-scale scientific and engineering project, such as the production of an aircraft, involves immense problems of interoperability for the many types of technological teams involved, as well as for the maintenance and adaptability over the project and product lifecycle. Major studies estimate that the costs to industry, of inadequate interoperability, are enormous. The Manufacturing Systems Integration Division at NIST (2005) argues that the next generation CAD/CAM/CAE software systems should meet the following critical requirements:

(R1) interoperability among software tools
(R2) collaboration among distributed designers and design teams
(R3) integration of data and knowledge across the product lifecycle
(R4) knowledge capture, exchange and reuse.

The product lifecycle involves an enormous range of information objects that emanate from and must be communicated between, very different knowledge frameworks – engineering, manufacturing, human resources, suppliers and customers.

In order to solve the above problems, we have created a new mathematical language that was published in the book, *A Generative Theory of Shape* (Springer-Verlag, 550 pages). This language fulfils the above four requirements in the following way:

We argue that there is one thing that all advanced engineering and business systems have in common – they each individually embody *intelligence* and *insight*. Thus, in the book, we ask the following question: What are the structural characteristics of an object that has been produced intelligently and insightfully? We argue that these structural characteristics form a language, and this language constitutes the only robust language that can satisfy requirements (R1)–(R4) mentioned above, i.e., because it is based on the only common factor of object-generation in all parts of the product lifecycle: intelligence and insight.

This language, proposed and elaborated in the book, constitutes new foundations to geometry – foundations that are shown to be the direct opposite of the standard foundations (Klein's Erlanger program and its associated mathematical systems). Throughout the present paper, the language and the book will both be referred to by the book's title *Generative Theory of Shape*. The fundamental difference between the standard foundations and the new foundations is that the new foundations characterise the information objects of any computational system.

To demonstrate the power of this geometry, the book shows that this single mathematical language formalises a large array of scientific and technical disciplines, including computer vision, general relativity, quantum mechanics, mechanical engineering, theory of differential equations, robotics, software engineering, the areas of human cognitive science, etc. Furthermore, in doing so, it discovers and exhibits fundamental correspondences between all these disciplines, thus allowing a single language to handle the multi-disciplinary nature of the information objects of the product lifecycle.

The present paper gives an introduction to some of the basic concepts, but tends to focus on the software engineering implications of the theory, which include defining a new type of object-oriented programming that far more deeply incorporates interoperability into class and object structure, by a new and powerful method of organising class specification, algebraically defining object creation and its relation to modification, predicting class discovery in software development and generatively structuring inheritance.

2   Intelligence and insight

The product lifecycle involves a vast array of scientific, engineering and business objects. We claim that these objects all have one thing in common: They have each been generated by intelligent and insightful action. Thus their only cross-disiplinary

interoperable representation is a structural characterisation of this fact. We therefore begin by stating what we regard as the two most basic properties of intelligent and insightful action:

- *Maximisation of transfer.* Any agent is regarded as displaying intelligence and insight when it is able to *transfer* actions used in previous situations to new situations. In fact, the agent must maximise the transfer of parts of generative sequences onto other parts of generative sequences. We can also regard transfer as corresponding to the notion of reusability. Thus, a basic part of our system is to give a *mathematical theory of reusability.*

- *Maximisation of recoverability.* Any intelligent agent must be able to infer the causes of its own current state, in order to identify why it failed or succeeded, and thereby edit its behaviour. Notice that it is part of a still larger problem, which we call the problem of recoverability: Given the present state of an object, recover the sequence of operations, which generated that current state. Thus a basic part of our system is to give a *mathematical theory of recoverability.*

Furthermore, when the two mathematical theories – of transfer and recoverability – are combined, they lead to a powerful mathematical structure, and it is this structure that provides a common language for scientific, engineering and business disciplines.

3   Transfer

Let us begin by giving an initial definition of what it means to maximise transfer:

Maximisation of transfer: *Make one part of a generative sequence a transfer of another part of the generative sequence, whenever possible.*

It will be argued that the appropriate formulation of this is as follows: A situation of transfer (see Figure 1) involves two levels: a *fiber group,* which is the group of actions to be transferred, and a *control group,* which is the group of actions that will transfer the fiber group. One can think of transfer as the control group moving the fiber group around some space; i.e., *transferring* it. The transferred versions of the fiber group are shown as the vertical copies in Figure 1 and will be called the *fiber-group copies.* The control group acts from above, and transfers the fiber-group copies onto each other, as indicated by the arrow.

Now let us begin to describe this structure in more detail. It will be argued that a situation of transfer is built out of two group actions. On the lower level, one has an action of a group $G(F)$, which we will call a *fiber group*, on a set F, which we will call a *fiber set*. On the upper level, one has an action of a group $G(C)$, which we will call a *control group*, on a set $C,$ which we will call a *control set*. Now, to model transfer, make the transferred versions of the fiber group, as follows: For each member c in the control set, make a copy $G(F)_c$ of the *fiber group $G(F)$*. These will be the transferred versions, and will be called the *fiber-group copies*, shown as the columns in Figure 1. Most crucially, the action of the control group $G(C)$ on the control set C can now be *imitated* by an action of the control group $G(C)$ on the collection of fiber-group copies. It is this imitating action that will be regarded as the transferring action that the control group has

on the fiber-group copies, i.e., this will be regarded as sending the fiber-group copies onto each other.

Figure 1 The control group transferring the fiber-group copies onto each other
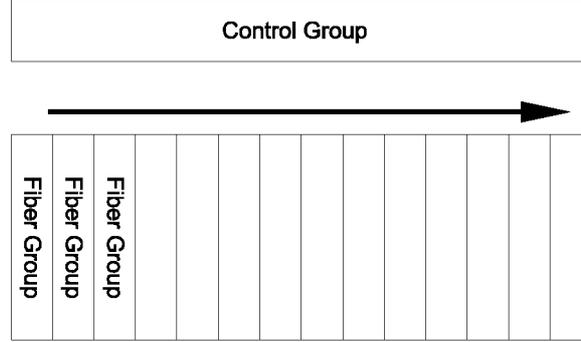
The next crucial thing we do is pull all these components together into a single encompassing structure. We argue that this encompassing structure is best given by a group-theoretic construct called a *wreath product,* which is defined as follows: Intuitively, a wreath product is a group that contains the entire structure shown in Figure 1. The structure of this total group is as follows: In Figure 1, the entire lower block shown is the direct product $\prod_{c \in C} G(F)_c$ of the fiber-group copies. Our theory calls this the *fiber-group product*. The wreath product group is then built up by adding, to the fiber-group product, the control group $G(C)$ by what is called a semi-direct product, explained thus: In any semi-direct product, the upper group (here the control group) sends the lower group (here the fiber-group product) to itself by rearrangements that preserve the latter's group structure. Such rearrangements are called automorphisms. In a wreath product, this automorphic action is one in which the control group sends the fiber-group copies onto each other in a way that exactly imitates the action of the control group on the control set. To state this rigorously, one constructs an automorphism representation,

$$\tau : G(C) \to \mathrm{Aut}\left\{\prod_{c \in C} G(F)_c\right\}$$

such that given an element *g* in the control group, its effect on the fiber-group product is this $\tau(g) : \prod_{c \in G(C)} G(F)_c \to \prod_{c \in G(C)} G(F)_{g^{-1}c.}$ Thus, it is from this automorphism representation, that one can construct the external semi-direct product $\left\{\prod_{c \in C} G(F)_c\right\} \circledS_\tau G(C),$ and it is this semi-direct product that one calls the wreath product of the fiber group and the control group, written in the following manner:

Fiber Group ⓦ Control Group = $G(F)$ ⓦ $G(C)$

$$= \left\{\prod_{c \in C} G(F)_c\right\} \circledS_\tau G(C). \tag{1}$$

Furthermore, given each *c* in the control set, let us call the set-theoretic Cartesian product $F \times \{c\}$ the corresponding *fiber-set copy*. Thus, call the union $F \times C$ of the fiber-set copies the *data set*. We then have a group action of the wreath-product group

$G(F) \,\circledW\, G(C)$ on the data set $F \times C$ as follows: Given an element in the wreath-product group, i.e., an ordered pair $\langle \phi \mid g \rangle$ where $\phi \in \prod_{c \in C} G(F)_c$, $g \in G(C)$ and given an element (f, c) in the data set, define the effect of the former element on the latter, thus:

$$\langle \phi \mid g \rangle (f, c) = (\phi(gc)f, gc) \in F \times C.$$

As an initial example, consider how the human visual system structures a square. In a sequence of psychological experiments (Leyton 1986b, 1986c), we showed that human vision represents a square *generatively,* in much the same way that one draws it on a sheet of paper – i.e., drawing the sides sequentially around the square. Notice that this in fact involves a crucial transfer structure described as follows: The first side is generated by starting with a corner point and applying translations to trace out the side, as shown in Figure 2.

Figure 2   The generation of a side, using translations
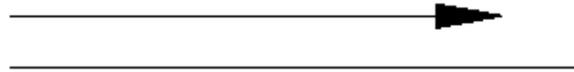
Next, this translational structure is *transferred* from one side to the next – rotationally around the square. In other words, there is transfer of translations by rotations. This is illustrated in Figure 3.

Figure 3   Transfer of translation by rotation
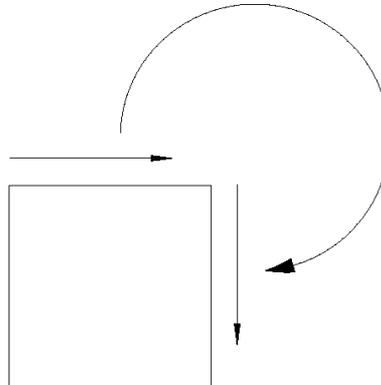
Therefore, the transfer structure is defined by the wreath product:

　　　Translations $\circledW$ Rotations

where Translations is the fiber group (corresponding to the side) and Rotations is the control group (transferring the side). This will now be defined rigorously, as follows:

The translation group (generating the side) will be denoted by the additive group $\mathbb{R}$. The rotation group is $\mathbb{Z}_4$, the cyclic group of order 4, which will be represented as

$$\mathbb{Z}_4 = \{e, \quad r_{90}, \quad r_{180}, \quad r_{270}\}$$

where r_θ means clockwise rotation by θ degrees. We now construct our wreath product of these two groups.

The group $\mathbb{Z}_4$ will be the control group, $G(C)$, and the control set will be the set C of four side-positions around the square:

$$c_1 = \text{top}, c_2 = \text{right}, c_3 = \text{bottom}, c_4 = \text{left}. \tag{2}$$

The control action of $\mathbb{Z}_4$ on the set $\{c_1, c_2, c_3, c_4\}$ will correspond to the clockwise rotation of the four side-positions onto each other.

The translation group $\mathbb{R}$ will be the fiber group, $G(F)$, and the fiber set will be the infinite line F containing the finite side as a subset. This is a crucial concept, as will be shown later. The fiber action of $\mathbb{R}$ on the fiber set F will be the obvious translation of the infinite line along itself.

Now since there are four elements in the control set $\{c_1, c_2, c_3, c_4\}$, there are four fiber-group copies $\mathbb{R}_{c_1}$, $\mathbb{R}_{c_2}$, $\mathbb{R}_{c_3}$, $\mathbb{R}_{c_4}$. Also, there are four fiber-set copies, which will be denoted as F_{c_1}, F_{c_2}, F_{c_3}, F_{c_4}. These will be the four infinite lines that contain the four finite sides as subsets. Each fiber-group copy (translation group) will act on its own 'personal' copy of the fiber set (infinite line).

One can now define the wreath product:

$$\mathbb{R}\,\text{Ⓦ}\,\mathbb{Z}_4 \tag{3}$$

where this group is the semi-direct product:

$$\left[\mathbb{R}_{c_1} \times \mathbb{R}_{c_2} \times \mathbb{R}_{c_3} \times \mathbb{R}_{c_4}\right] \text{Ⓢ}_\tau \mathbb{Z}_4. \tag{4}$$

As indicated by τ, the automorphic action of the control group $\mathbb{Z}_4$, on the fiber-group product $\mathbb{R}_{c_1} \times \mathbb{R}_{c_2} \times \mathbb{R}_{c_3} \times \mathbb{R}_{c_4}$, corresponds to the action of $\mathbb{Z}_4$ on the control set $\{c_1, c_2, c_3, c_4\}$. This means the fiber-group copies are rotated around the square.

The data set $F \times C$, in this example, is given by the four infinite lines containing the four finite sides. We can think of this as *four infinite wires* overlapping each other.[1]

In our generative theory, the four infinite wires are cut down to their visible portion, the finite square, by placing what we call an *occupancy group*, $\mathbb{Z}_2$ (a cyclic group of order 2), at each point along the infinite line containing a side. The group switches between two states, 'occupied' and 'non-occupied,' and is wreath sub-appended below the above group thus

$$\mathbb{Z}_2\,\text{Ⓦ}\,\mathbb{R}\,\text{Ⓦ}\,\mathbb{Z}_4$$

For ease of reading, the occupancy level will often be left out of the notation, when not needed in the immediate discussion.

Now observe that the group at expression (3) gives *generative coordinates* to the square, as follows: We can identify the members, c_i, of the control set with the members r_θ of the control group. Thus, any fiber-group copy can be labelled $\mathbb{R}_{r_\theta}$ and its elements can be labeled t_{r_θ}. Therefore, any point on the square can be described by a pair of coordinates:

$$(t, r_\theta) = t_{r_\theta} \in \mathbb{R}_{r_\theta}.$$

The first coordinate gives the generative (translational) distance along a side and the second coordinate gives the generative (rotational) distance of a side from the first generated side. Therefore, a point is given a complete generative description from the

starting point that *maximises transfer.* The transfer structure on coordinates is illustrated in Figure 4.

Figure 4   The transfer structure given to the coordinates
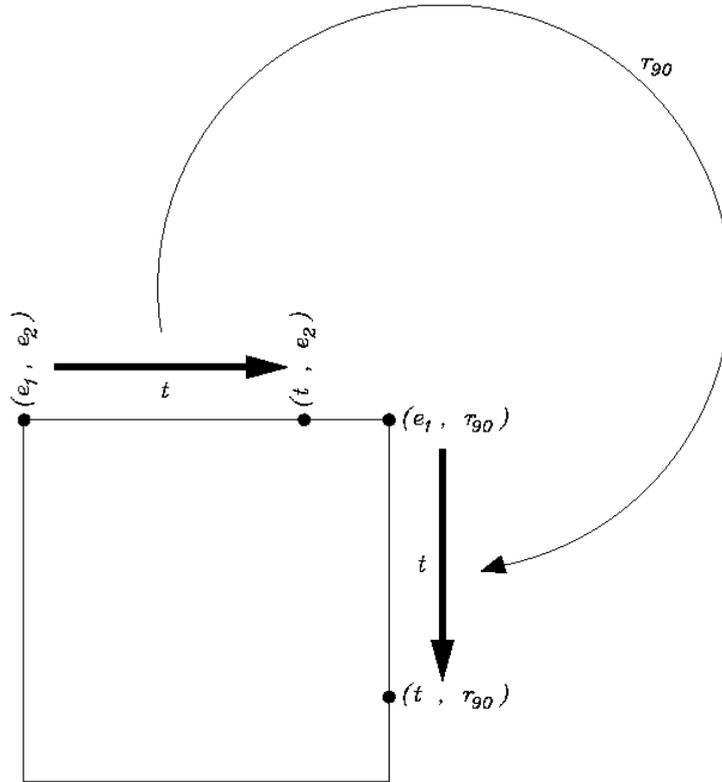
Now, *deformed* shapes are handled in our system by adding extra layers of transfer. For example, to obtain a parallelogram, one adds the group of linear transformations, $GL(2, \mathbb{R})$, onto the two-level group of the square in the following manner:

$$\mathbb{R} \, \textcircled{w} \mathbb{Z}_4 \textcircled{w} GL(2, \mathbb{R}). \tag{5}$$

Notice that the operation used to add $GL(2, \mathbb{R})$ on to the lower structure $\mathbb{R}\textcircled{w}\mathbb{Z}_4$ is, once again, the wreath product $\textcircled{w}$ which means that $GL(2, \mathbb{R})$ acts by *transferring* $\mathbb{R}\textcircled{w}\mathbb{Z}_4$, as follows: Since the fiber group $\mathbb{R}\textcircled{w}\mathbb{Z}_4$ represents the structure of the square, this means that $GL(2, \mathbb{R})$ transfers the structure of the square onto the parallelogram. However, we saw that the structure $\mathbb{R}\textcircled{w}\mathbb{Z}_4$ of the square is itself a transfer structure, where translations are transferred by rotations. This transfer structure is itself transferred, by $GL(2, \mathbb{R})$, onto the parallelogram. That is, the transfer structure in Figure 4 is transferred onto Figure 5. Therefore, we have *transfer of transfer.* This recursive transfer is encoded by the successive $\textcircled{w}$ operations in expression (5).
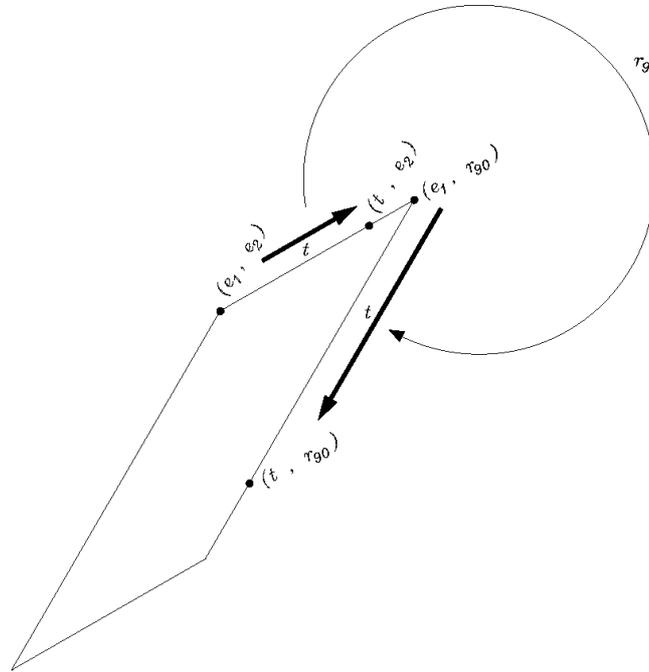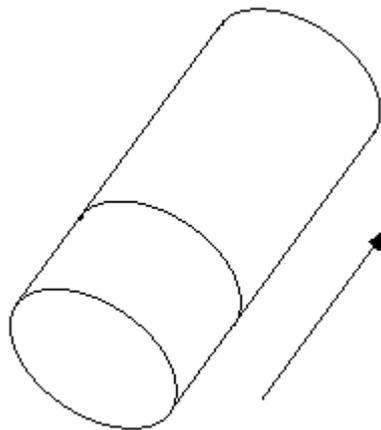
Figure 5 The transfer of transfer

Figure 6 The sweep structure of a cylinder

What has been illustrated here is our principle of the *maximisation of transfer*. The parallelogram is given a generative description, all the way up from a point, that maximises transfer.

The theory is equally applicable to 3-dimensional shape. For example, consider the structure of a cylinder. In computer vision and graphics, cylinders are described generatively as the sweeping of the circular cross-section along the axis, as shown in Figure 6. To our knowledge, the group of this sweeping structure has never been given. We propose that the appropriate group is:

$$SO(2) \;\textcircled{w}\; \mathbb{R} \qquad\qquad\qquad\qquad\qquad (6)$$

where $SO(2)$ is the rotation group generating the cross-section, and $\mathbb{R}$ is the translation group along the axis. The wreath product symbol $\textcircled{w}$ means that the translation group $\mathbb{R}$ *transfers* the rotation group $SO(2)$ along the axis.

4   Implicit and explicit notation

We have used two notations for a wreath product, which were illustrated with the example of the square: The first is

   Implicit Notation: $\mathbb{R}\textcircled{w}\mathbb{Z}_4$.

That is, although this group contains four copies of the fiber $\mathbb{R}$, they are not explicitly shown in the notation. In contrast, the following notation, for the same wreath product group, exhibits more of the structure involved:

   Explicit Notation: $\left[\mathbb{R}_{c_1} \times \mathbb{R}_{c_2} \times \mathbb{R}_{c_3} \times \mathbb{R}_{c_4} \right] \textcircled{s}_\tau \mathbb{Z}_4$.

This notation reveals the following important information:

- it shows that there are four fiber-group copies

- it shows that these fiber-group copies are glued together via the direct product operation $\times$, as are the fiber-group copies in any wreath product

- it shows that the control group $\mathbb{Z}_4$ is glued onto the fiber-group product via a semi-direct product $\textcircled{s}$, as is the case in any wreath product

- it shows that the particular automorphic action τ, which the control group has on the fiber-group product, is that of sending the fiber-group copies onto each other in a way that corresponds to the action of the control group on the control set. The latter action would have to be giv   en in a separate annotation.

We will now use our transfer-based theory of shape to understand the two most popular ways of drawing a square: The first is the sequential drawing of the sides around the square, which corresponds to the transfer structure discussed above. The other most popular method is this: First draw the top side followed by the bottom side, both from left to right; and then draw the left side followed by the right side, both from top to bottom.

   What we will show is that this scenario is also chosen to maximise transfer. In order to understand what is going on in this scenario, we must consider the reflectional structure of the square. Figure 7 shows three of the reflection axes of the square. They are reflection m_V about the vertical axis, reflection m_H about the horizontal axis and reflection m_D about one of the diagonal axes. The important thing to observe is this: The diagonal reflection *transfers* the vertical reflection onto the horizontal reflection.

Figure 7 Three of the four reflection axes of a square
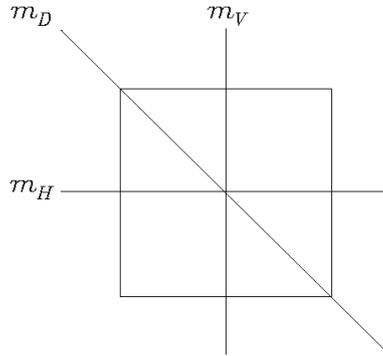
This transfer structure is critical. It allows us to regard the vertical reflection and the horizontal reflection as *fibers* and the diagonal reflection as *control* that sends the two fibers onto each other.

To fully define this, let us form the reflection groups for each of the three axes. The reflection group for the vertical axis is $\mathbb{Z}_2^V = \{e, \quad m_V\}$, which consists of *e*, the null-transformation and m_V, the vertical reflection. Correspondingly, the reflection group of the horizontal axis is $\mathbb{Z}_2^H = \{e, \quad m_H\}$ and the reflection group for the diagonal axis is $\mathbb{Z}_2^D = \{e, \quad m_D\}$.

We now simply create a wreath product, in which the diagonal reflection group is the control group, and the two fibers are the vertical group and the horizontal group. In the explicit notation, this wreath product is written in the following manner:

$$\left[\mathbb{Z}_2^V \times \mathbb{Z}_2^H \right] \textcircled{S}_\tau \mathbb{Z}_2^D . \tag{7}$$

This is read in the following way: The control group $\mathbb{Z}_2^D$, to the right of the symbol $\textcircled{S}$, interchanges the two fibers $\mathbb{Z}_2^V$ and $\mathbb{Z}_2^H$ to the left of the symbol $\textcircled{S}$; i.e., transfers the two fibers onto each other. In the implicit notation, the group is written as follows:

$$\mathbb{Z}_2 \textcircled{W} \mathbb{Z}_2.$$

Finally, below this group, one needs to add the level $\mathbb{R}$, which draws a side. Thus, one obtains the following wreath product:

$$\mathbb{R}\textcircled{W}\left[\mathbb{Z}_2^V \times \mathbb{Z}_2^H \right] \textcircled{S}_\tau \mathbb{Z}_2^D = \mathbb{R}\textcircled{W}\mathbb{Z}_2 \textcircled{W} \mathbb{Z}_2. \tag{8}$$

This shows that the second most popular scenario for drawing a square is a three-level transfer hierarchy: One draws the top side and transfers this *reflectionally* to create the bottom side; and finally one takes the entire top–bottom reflectional scenario, just given, and *reflects* it about the diagonal to obtain the *reflectional* scenario that draws the left and right sides.

5   Object-linked inheritance

The term *inheritance,* in object-oriented programming, refers to the passing of properties from a parent to a child, (Meyer, 1997). The child takes on these parent properties, but also adds its own. The type of inheritance that is discussed in all books on object-oriented programming is *class* inheritance, which is an *abstraction* hierarchy. Our mathematical theory of this will be elaborated in Sections 14–18.

At this stage, we will give a theory of a different form of inheritance that is also used prolifically in all components of the product lifecycle from CAD/CAM/CAE to ERP/SCM/CRM. This type of structure is not an abstraction hierarchy, but a hierarchy in which objects are defined by dependencies that are associative references to other objects in the hierarchy. For example, in CAD, a model can be given by a graph of its constituents in which the parent–child links determine which objects must be regenerated when the user decides to change some selected object. That is, the alteration of a property of the selected object (a parent) will necessitate changes in the properties of other objects (its children). This type of parent–child dependency occurs in all components of the CAD/CAM/CAE cycle – part design, assembly, machining, etc. Furthermore, it occurs across components, e.g., machining data can be attached as children to a face in the CAD model. Also, it occurs within business applications, e.g., as Allen (2000) points out, the associative parent–child relations in the CAD/CAM domain are analogous to standard behaviours in the spreadsheets associated with business objects. Furthermore, it occurs between engineering and business applications, e.g., the bill of material (BOM) appearing within a CAD drawing.

In the present paper, the phrase *object-linked inheritance* will be used to refer to the above fundamental type of parent–child dependency. *Generative Theory of Shape* gives the following theory of object-linked inheritance:

Algebraic theory of object-linked inheritance: *Object-linked inheritance arises from a wreath product:*

$$Child \; ⓦ \; Parent$$

where Child is the command group of the child, and Parent is the command group of the parent. Thus for a set of n objects that are linked hierarchically from Object 1, the ultimate child, up to Object n, the ultimate parent, the complete transform group of the hierarchy is given by

$$G_1 \; ⓦ \; G_2 \; ⓦ \; \cdots \; ⓦ \; G_n$$

where G_i is the personal transform group of Object i.

Let us consider some illustrations:

6   Kinematics

As an example of the object-linked inheritance, let us consider parent–child structure in kinematics. Inheritance of this type is basic to all engineering representations of mechanisms – in assembly, machining, animation, robotics, etc. We now propose the following algebraic theory of this kind of inheritance:

Algebraic theory of kinematic inheritance: *Consider a set of $n + 1$ objects: Object O_1 to O_n and the World. Let each object O_i be created with a frame F_i and, for each $i < n$, let object O_i be linked to object O_{i+1} such that its personal transform group $_{F_i}G_i^{F_{i+1}}$ relates its frame to the frame of its parent; and let the frame of object O_n be related via its transform group $_{F_i}G_n^W$ to the world frame W. Then the group of the entire transform structure is the wreath product:*

$$_{F_1}G_1^{F_2} \,\textcircled{w}\, _{F_2}G_2^{F_3} \,\textcircled{w}\, \cdots \,\textcircled{w}\, _{F_n}G_n^W \,.$$

It will now be shown that this gives a deep understanding of the rationale of using relative motion decomposition in kinematics. First note that it is well accepted that such decomposition is a choice made by the designer/engineer/physicist in order to convert a complex motion into a comprehensible form. However, what has not been understood is what this really means, both conceptually and algebraically. Our mathematical theory of intelligence and insight gives the following understanding of the rationale of relative motion decomposition:

Algebraic theory of relative motion: *Relative motion follows from the principle of the maximisation of transfer. To obtain a relative motion representation, apply the following rule:*

Decompose the motion into two symmetry groups, such that one group transfers *the other.*
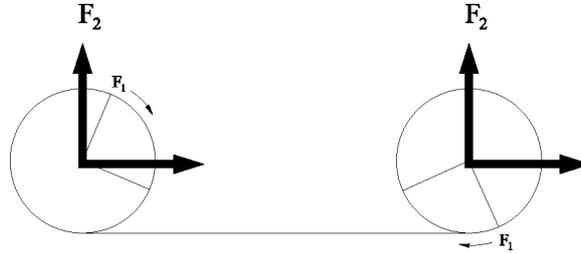
This results in the following wreath product:

relative motion $\textcircled{w}$ absolute motion.

To illustrate, let us begin with the classical kinematic example of a cycloid. This is standardly represented by a relative motion decomposition, in which the components are a circle and a straight line, with the former rolling along the latter and the tracing point being on the circle. The procedure accords with the above theory: The motion is decomposed into two symmetry groups, such that one *transfers* the other. The two symmetry groups are $\mathbb{R}$, the translation group of the circle-centre parallel to the line and *SO*(2), the rotation group of the tracing point around the circle-centre. Because the translation group *transfers* the rotation group, the motion is structured by the following wreath product:

SO(2) $\textcircled{w}$ $\mathbb{R}$.

To understand the corresponding inheritance structure, consider the frames involved, as illustrated in Figure 8. First, there is frame F_1, whose origin is the circle-centre, and which is fixed relative to the circle, therefore rotating with the circle. Second, there is frame F_2, whose origin is also the circle centre, but whose orientation is fixed relative to the straight line, and which therefore translates with the circle along the line.

Figure 8   Frames assigned in the cycloid example

The crucial inheritance structure is this: The rotating frame F_1 *inherits* the translation of frame F_2. Therefore, according to our algebraic theory, the inheritance structure is given by the following wreath product:

$$_{F_1} SO(2)^{F_2} \; \textcircled{w} \, _{F_2} \mathbb{R}^W . \tag{9}$$

To understand additional aspects of our algebraic theory of inheritance, let us apply the theory to *animation.* Consider a solar system consisting of the earth revolving around the sun and the moon revolving around the earth. The obvious parent-to-child order is sun to earth to moon.

Now, the standard practice of animators is to use two frames centred within each parent planet:

(1)   a frame fixed within the planet

(2)   a dummy frame (sharing the same centre) that rotates relative to frame (1), but
      together with the child that rotates around the planet.

Although frame (2) is centred in the parent planet, it actually 'belongs' to the child planet in the sense that it carries the orbiting relative motion of the child with respect to the parent.

The total number of frames in the above planetary system is therefore six, and they are as follows: w = world; s = sun; sd = sun-dummy; e = earth; ed = earth-dummy; m = moon. These six objects are completely ordered in a kinematic inheritance hierarchy along the list. That is, each object inherits the transform of the object preceding it.

Using our theory of kinematic inheritance, the complete transform structure for the animation is given by the following wreath product:

$$_m \mathbb{R}^{ed} \; \textcircled{w} \, _{ed} SO(2)^e \; \textcircled{w} \, _e \mathbb{R}^{sd} \; \textcircled{w} \, _{sd} SO(2)^s \; \textcircled{w} \, _s \mathbb{R}^w . \tag{10}$$

To understand this structure, observe that there are two groups that are used: $\mathbb{R}$ the translation group along a straight line and $SO(2)$ the rotation group in the plane. These two groups alternate successively along the sequence. The six successive frame objects are given as the successive indexes. The important point is that each wreath-product symbol $\textcircled{w}$, along the sequence, indicates that the group to the right of the symbol transfers the group to the left.

Notice the editability property of the hierarchy accords exactly with what one requires from parametric modelling. For example, editing the radial value in $_e \mathbb{R}^{sd}$ will propagate downward to affect the motion of the children, but not upward to affect the motion of the parents.

Applications

Generative Theory of Shape gives many more applications of the above theory of relative motion in the fields of robotics, classical and quantum mechanics, perception, etc. For example, in robotics, the above theory is applied to give several levels of analysis for robot manipulators, elaborating a new algebraic formulation of their kinematics, explaining their frame assignment, etc.; in classical and quantum mechanics, the theory is applied to explain the decomposition theorems concerning linear and angular momentum, as well as the redescription of a two-particle system in terms of fictitous masses – the total mass and the reduced mass, etc.; in perception the above theory is applied to solve the main Gestalt motion phenomena – induced motion, separation of systems, the Johannson relative/absolute motion effects, etc. – thus explaining the cognitive aspects of the human user.

Comment on differential equations and dynamics

Differential equations are by far the most frequently used modelling method throughout the world. They are used prolifically throughout all aspects of the product lifecycle – from the definition and analysis of engineering objects, to facilities planning and maintenance, to operations support. Clearly, all this depends on methods for *solving* differential equations, and a large variety of such methods have been developed. However, these methods arise out of the approach invented by Sophus Lie, by formulating the machinery of Lie groups and Lie algebras. *Generative Theory of Shape* shows how to reformulate the Lie theory in terms of the different mathematics of our generative theory. Our reformulation contains advantages in that its mathematics represents the stages of scientific discovery, e.g., induction across scientific experiments, and therefore knowledge capture.

Another significant result of our system is that it gives a new formulation of the conservation laws of physics, which of course are basic to all the diverse engineering models of the product structure. For example, Chapter 20 of the book illustrates our new formulation with respect to quantum mechanics, including an analysis of the Galilean Lie algebra, its non-solvability, the role of Racah's theorem in providing Casimir operators whose eigenvalues characterise the multiplets of semisimple Lie groups, etc. Since our method is applicable to all conservation laws, it goes significantly towards giving a unified language for knowledge capture across the diverse engineering models of PLM.

7   Parametric structure and object-orientedness

Standardly, the parametric structure of a feature-based design is described in object-oriented terms in two ways:

- dependencies between objects are given by associative inheritance

- the relation of a parameterised family to its created examples is seen as analogous to the class–instance relation.

In Sections 5 and 6, we saw that *Generative Theory of Shape* gives an algebraic theory of the first type of structure in terms of a wreath product between the command groups of objects. Later, we shall see that the generative theory gives an algebraic theory of the second type of structure (in fact, class structure generally) in terms of a wreath product

between the command group and the creation procedure/invariance clauses within a particular class. Thus, the power of our theory is that it formulates both types of structure with the same conceptual frame-work, transfer, and the same algebraic language, wreath products, and makes the first kind of structure come from a transfer relation (wreath product) *between* objects, and the second kind of structure come from a transfer relation (wreath product) *within* an object class.

8   Feature editing and persistent naming

As is well known, feature editing requires the use of persistent names to preserve the associative parent–child relationships. Persistent names show that newly appearing objects should be considered new versions of old ones. *Generative Theory of Shape* gives a solution to this problem in a particularly elegant way, as follows: Describing a new object as a new version of an old one is to describe it as the *transfer* of the old one. Our system, therefore, represents the two objects as fiber copies within a wreath product. Furthermore, most crucially, wreath products have a powerful algebraic machinery, which includes a persistent naming set automatically: the control set.

As an example, consider the case in which a square has been modified to become a parallelogram, as in the change from Figures 4 and 5. Now, by our system, this is a case of the *transfer* of a square onto a parallelogram and is therefore given by the wreath product $\mathbb{R} \mathbin{ⓦ} \mathbb{Z}_4 \mathbin{ⓦ} GL(2,\mathbb{R})$. Furthermore, recall that the control set in the wreath product $\mathbb{R} \mathbin{ⓦ} \mathbb{Z}_4$ of the square is

$$c_1 = \text{top}, \quad c_2 = \text{right}, \quad c_3 = \text{bottom}, \quad c_4 = \text{left}.$$

This is also transferred onto the parallelogram. Thus, for example, the label c_2, for the right side of the square, is transferred onto the newly created right side of the parallelogram, even though the former side is shorter and vertical and the latter side is longer and slanting. Most crucially, the *structure* of the right side of the square is transferred onto the *structure* of the right side of the parallelogram. Therefore, any child application associated with this side, e.g., machining data, will be correspondingly transferred with the appropriate modification.

Generally, as summarised in the footnote, the persistent naming issue is handled in our system in a significantly different way from that developed in the paper of Capoyleas et al. (1996).[2]

9   Recoverability

Recall that our theory is founded on *two* principles – the maximisation of transfer and the maximisation of recoverability. The preceding sections have been dealing with transfer, and we now turn to recoverability. Furthermore, transfer and recoverability will be shown to come together in a powerful mathematical structure.

By *recovery,* we mean the following problem:

Given a data set, recover or infer a sequence of operations that generate the set.

Our first book (Leyton, 1992) was a 630-page analysis of this problem, and one of the main conclusions of this analysis was the following principle.

Asymmetry principle: *The only recoverable operations are symmetry-breaking ones. That is, a generative sequence is recoverable only if it is symmetry-breaking on each of the successively generated states.*

Now it is clear that there are many processes in the world that are not symmetry-breaking, but are symmetry-increasing, e.g., a tank of gas settling to equilibrium under the standard entropy-increasing process. Our theory says the following:

Symmetry-increasing processes: *A symmetry-increasing process is recoverable only if it is symmetry-decreasing on successive data sets.*

So, for example, you can recover the fact that the tank of gas was entropy-increasing over time, if you kept a set of records (e.g., photographs) and the records are linearly ordered, e.g., they are laid out from left to right on a table, in which case, the sequence of photographs breaks the left–right symmetry of the table. That is, the increase in spatial symmetry in the tank of gas is made to correspond to a decrease in spatial symmetry of the record structure.

Before elaborating the rules of recoverability, it is valuable to show how recoverability combines with transfer.

10 Combining transfer and recoverability

A basic factor emerges from the above discussion: in order to ensure recoverability, the control group must be symmetry-breaking on its fiber. Generally, the following should be observed:

- the transfer component of our theory leads to wreath products

- the recoverability component adds the condition that the control group must be symmetry-breaking on its fiber

- thus the combination of transfer and recoverability leads to a powerful mathematical structure, we call *symmetry-breaking wreath products.*

11 New theory of symmetry-breaking

Close examination reveals that this gives a far more powerful theory of symmetry-breaking than the conventional one that underlies physics and chemistry.

Conventional view of symmetry-breaking: *Symmetry-breaking is a reduction of symmetry group.*

As an example, consider the transition from a square to a parallelogram, which is a symmetry-breaking one. The conventional view says that the symmetry group of a square is D_4, which consists of the eight Euclidean transformations that map the square to itself:

four rotations and four reflections. In contrast, a parallelogram is given by the symmetry group $\mathbb{Z}_2$, which consists of the only two Euclidean transformations that map a parallelogram to itself – rotation by 0° and rotation by 180°. Thus, the transition from a square to a parallelogram is given by the following transition of groups:

$$D_4 \rightarrow \mathbb{Z}_2. \tag{11}$$

In fact, the group $\mathbb{Z}_2$ is a subgroup of D_4, which means that the transition is given by reduction of symmetry group.

This view of symmetry-breaking has dominated physics and chemistry for nearly a century. However, according to our theory, this is inherently weak because it means a loss of algebraic structure. That is, in the conventional view, as one goes from a simpler object (such as a square) to a more complex object (such as a parallelogram), one is reducing the size of the description, which is logically absurd.

In contrast, Section 3 illustrated our theory of symmetry-breaking using the same example, the transition from a square to a parallelogram. We modelled this transition as follows: one takes the symmetry group of the square, which we gave as $\mathbb{R}\,Ⓦ\,\mathbb{Z}_4$ and *adds* the group of linear transformations, *via a wreath product:*

$$\mathbb{R}\,Ⓦ\,\mathbb{Z}_4 \rightarrow \mathbb{R}\,Ⓦ\,\mathbb{Z}_4\,Ⓦ\,GL(2,\mathbb{R}). \tag{12}$$

Therefore, in our approach, symmetry-breaking actually preserves the original group, and in fact increases it. The enormous power of this approach will be seen shortly, but first let us state it precisely:

New view of symmetry-breaking: *The breaking of a symmetry group G_1 is given by its extension by another group G_2 via a wreath product thus: $G_1\,Ⓦ\,G_2$, where G_2 is the symmetry group of the asymmetrising action.*

The first powerful feature of this approach is as follows: Notice that the wreath product symbol $Ⓦ$ in the above statement implies that the *past symmetry G_1 is transferred onto the present broken symmetry.*

Let us give an initial illustrating example. Consider a bent pipe that one might come across in the street. Merely by the fact that one understands it as a bent pipe means that one sees the past symmetric version as transferred onto the present asymmetric version.

12　Memory storage

The recoverability, from an object, of the operations that generated it, means that the object acts as a *memory store* for the operations. As a result of the recoverability condition, *Generative Theory of Shape* gives new foundations to geometry that are fundamentally different from the standard foundations, which are based on Klein's Erlanger program, which says that a geometric object is an invariant of action. We argue that, if an object is invariant under action, then the action is not recoverable from the object; i.e., the object is memoryless with respect to the action. Since the standard foundations for geometry try to maximise invariants, they concern the maximisation of memorylessness. In contrast, our new foundations try to maximise the memory storage.

Objects: *Whereas the objects of the standard foundations for geometry are memoryless objects (i.e., invariants), the objects of our new foundations for geometry are memory stores.*

Furthermore, our new foundations for geometry show the structure of any memory store:

Fundamental structure of memory store: *Any memory store is structured as a symmetry-breaking wreath product.*

This gives a fundamental constraint on the structure of the scientific, engineering and business objects of the product lifecycle. The integration of the heterogenous lifecycle components requires that the objects of those components, whether CAD designs, machined parts or bills of material, etc., allow those objects to be *records* of the actions that produced them. The above principle shows how they can be structured such that they can act as maximal records of those actions. The remainder of this paper examines this in more detail.

13 External vs. internal inference

This section starts to elaborate, in more detail, our theory of recoverability of generative history. First it is necessary to distinguish between two types of inference.

Definition 1: In *external inference*, also called the *single-state assumption*, the observer assumes that the data set contains a record of only a single state of the generative process (i.e., a single snap-shot). Any inferred preceding state therefore does not have a record in the data set. We say that it is external to the data set.

An example of external inference is where one is presented with an object only in a deformed state (e.g., a bent pipe), and infers the deformation (e.g., bending) that was applied to it.

Definition 2: In *internal inference*, also called the *multiple-state assumption*, the observer assumes that the data set contains records of multiple states of the generative process (i.e., multiple snap-shots taken over time). A state, recorded in the data set, can therefore have a past state that also has a record internal to the data set.

An example of internal inference is where one is presented with a trace of states, e.g., tracks in snow, and infers the sequence of movements that produced that trace. We shall often use the phrase *internal structure* and *trace structure* interchangeably.

Remarkably, although external and internal inference apply to very different types of situations, e.g., deformation vs. traces, they are both carried out by using the Asymmetry Principle. What is different is the nature of the asymmetries involved:

Application of rules: *In external inference, the Asymmetry Principle is applied to* intra-record *asymmetries. In internal inference, the Asymmetry Principle is applied to* inter-record *asymmetries.*

A rule that turns out to be fundamental to the entire process of recovering generative history will now be proposed.

Externalisation principle: *To maximise recoverability, any generative sequence, inferred by external inference, must lead back to a starting state whose internal structure corresponds to an iso-regular group.*

where we define:

Iso-regular group: *This is a group satisfying the following three conditions:*

- *it is an n-fold wreath product, $G_1 \textcircled{W} G_2 \textcircled{W} \cdots \textcircled{W} G_n$; i.e., a structure of hierarchical transfer*

- *each level, G_i, is generated by a single generator (i.e., it is either a cyclic group or a connected 1-parameter Lie group)*

- *each level, G_i, is an isometry group on its space of action.*

To begin to explain this principle, let us start with very simple examples. Notice first that two of the groups given earlier are examples of iso-regular groups:

   Square: $\mathbb{R} \textcircled{W} \mathbb{Z}_4$

   Cylinder: $SO(2) \textcircled{W} \mathbb{R}$.

That is, each of the levels can be generated by a single generator and is used as an isometry group on its space of action.

   Having given an illustration of an iso-regular group, let us now turn to the Externalisation Principle itself. It says that external inference will lead ultimately back to a past state that is given by an iso-regular group. As an illustration of this, it is worth considering the following results:

   In a series of psychological experiments (Leyton, 1986b, 1986c), we found that when subjects are presented with a parallelogram oriented in the picture plane as shown in Figure 9(a), they see it as a rotated version of the parallelogram in Figure 9(b), which they then see as a sheared version of the rectangle in Figure 9(c), which they then see as a stretched version of the square in Figure 9(d). The remarkable thing is that the only data that the subjects are actually given is the first figure, the rotated parallelogram. That is, inference is *external*; i.e., the inferred past states are external to the presented data – the rotated parallelogram. The experiments found that on being presented with this figure, their minds successively produce the entire sequence shown.

Figure 9   Psychological results

   (a)   (b)   (c)   (d)
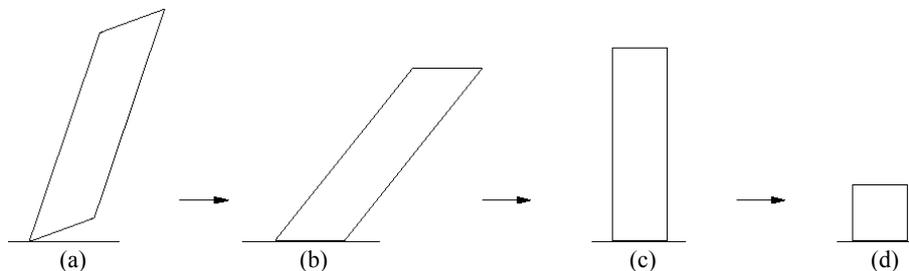
   Source: Leyton's (1986b, 1986c)

What the subjects appear to be doing, in this experiment, is inferring the generative history of the presented figure, the rotated parallelogram. That is, they are saying that, in the past, this figure started out as a square (the extreme right), which was then stretched to produce the rectangle, which was then sheared to produce the parallelogram, which was then rotated to produce the rotated parallelogram. That is, the sequence *from right to left* is the generative sequence, forward in time, from the past to the present. Thus, in producing the sequence *from left to right,* the subjects are running time backwards from the presented shape.

Now observe the following: In recovering the successive shapes, backwards in time, i.e., from the rotated parallelogram to the square, the subjects are using the Asymmetry Principle, as follows: The presented figure, the rotated parallelogram, has three asymmetries (distinguishabilities), which are as follows:

- the distinguishability between the orientation of the parallelogram and the orientation of the environment, as indicated for example by the distinguishability between the orientation of the bottom side of the parallelogram and the ground line

- the distinguishability in size between adjacent angles

- the distinguishability in length between adjacent sides.

It is clear that what the subjects are doing, in producing the sequence from left to right, is removing these three distinguishabilities successively, backward in time. In other words, they are recovering the past, by using the Asymmetry Principle, i.e., converting asymmetries to symmetries, backwards in time. This means that, when one considers the sequence from right to left, one concludes that *in the forward-time direction, the sequence is symmetry-breaking.*

The above experiments are an example of the successive reference phenomenon we discovered in the 1980s and published in Leyton (1986a, 1986b, 1986c). We modelled this phenomenon by a dynamical system on the group hierarchy, such that each level has a flow from any group element back to the identity element on that level, and there is a precedence ordering such that the flow on any level moves to its identity element before the flow on any of its fiber levels. We called such a dynamical system, an *algebraic stability ordering.*

Now let us return to the theory of recoverability. The next thing we should observe is that the recovered history in Figure 9 accords with our Externalisation Principle: As pointed out before, the three successive uses of the Asymmetry Principle infer a succession of past states that are external to the initially presented shape (the rotated parallelogram). Therefore, the recovered sequence (from left to right) is achieved by external inference. Now, this inference goes back to a square, which, we have seen, has an internal structure given by an iso-regular group. Therefore, the sequence conforms to the Externalisation Principle, which says that external inference always leads back to an internal structure given by an iso-regular group.

As another example of the Externalisation Principle, consider a bent pipe. It is clear that merely by describing this as bent, one understands the generative origin to have been a straight pipe, i.e., a cylinder. But we saw that a cylinder has an internal structure (trace structure) given by an iso-regular group. Therefore this inference also conforms to the Externalisation Principle.

Let us now understand the relationship between the Externalisation Principle and the Asymmetry Principle, which implies that asymmetries in the data set must go back to symmetries in the previously generated states. Recall also that we distinguished between two realisations of the Asymmetry Principle: external inference, which goes back to a past state outside the data set, and internal inference, which goes back to a past state inside the data set. The Externalisation Principle concerns the first of these two classes of inference. *The Externalisation Principle is the Asymmetry Principle in half of all possible cases: those of external inference.* It states that the ultimate initial states that are recoverable by external inference are those that are given by iso-regular groups. This places an enormously strong condition on initial states. The forward-time statement of this principle is as follows: *Any history recoverable by external inference must break the symmetry of an iso-regular group.*

Observe that the principle shows what knowledge needs to be retained and what can be inferred. Consider, for example, the interoperatibility problem. Currently, there is a new development in setting the ISO standard for the transmission of designs between different CAD programs. It is being understood that an important representation for such transmission is the history of design operations used to create the transmitted product model; see Pratt (2004). This accords with the theory advocated in our papers and books for the last 20 years, that shape is equivalent to the generative operations that produced it (Leyton, 1986a, 1986b, 1986c, 1988, 1989, 1992, 2001). However, our generative theory also adds the following principle, that we argue is crucial: It says that generativity is not enough, but that *recoverability* is needed. With our mathematical theory of recoverability, the receiving system can actually *infer* the history of the design operations; that is, recoverable operations need not be transmitted. Thus our first book (Leyton, 1992) gave a 630-page rule-system for the inference of generative operations. Furthermore, recoverability determines the best way to actually represent the generative operations. In particular, the Externalisation Principle is one of the main inference rules and determines that the generative operations, recoverable by external inference, are best described by symmetry-breaking wreath products founded on iso-regular groups as fibers.

Sections 14–18 will show that, because of this, the Externalisation Principle determines the most intelligent way to elaborate class hierarchies, and to organise class specification, in object-oriented programs. This will lead us to propose a new type of object-oriented programming that is much more rigorous than the current types.

The multi-disciplinary power of the Externalisation Principle

Generative Theory of Shape shows that the Externalisation Principle is a fundamental law of organisation, independent of discipline. For example, we showed the following:

- the principle gives a systematic elaboration and classification of the shape primitives of CAD

- it explains procedures for recognition of machining features, e.g., it gives a mathematical understanding of the hint-based recognition system of Vandenbrande and Requicha (1993)

- it determines the inverse-optics structure of computer vision, showing how the projective geometry must be organised to ensure recoverability of the imaging environment

- it determines the gravitational relation between the differential geometries of general relativity and special relativity

- it explains particle organisation in quantum mechanics, e.g., the relation between the principal Hamiltonian of the hydrogen atom, the fine splitting, hyperfine splitting and Zeeman effects.

The domain-independent validity of the Externalisation Principle therefore makes it a powerful tool for the integration of the multidisciplinary representations that occur in the product lifecycle.

Comment on control theory

Control theory is fundamental to the engineering models of a product. We have seen that *Generative Theory of Shape* gives a new algebraic theory of both kinematic structuring, e.g., decomposition into relative motion systems, as well as dynamical laws, e.g., the setting up and solution of differential equations, and the conservation laws of classical and quantum mechanics. However, the theory also affords a new mathematical approach to *control-theoretic* aspects. As an example, our concept of an algebraic stability ordering, elaborated in Leyton's (1986a, 1986b, 1986c), has lead Kirupaharan and Dayawansa (2001) to develop an entirely new type of mathematical control theory for posture control. Kirupaharan and Dayawansa define the posture control problem (e.g., for a robot) as a hierarchy of successively embedded posture control problems $P_1, P_2, \ldots, P_n$, in which the solution of each problem P_i, formulated in terms of the stabilisation of its Euler–Lagrange equations, is used for the solution of problem P_{i+1}. Thus, one obtains a hierarchy of stabilisation problems, which corresponds to the successive stabilisation structure given by our algebraic stability ordering. Therefore, our theory has lead to new forms of mathematical control theory that are very different from classical and modern control methodologies.

14 Class inheritance

The theory presented in this paper allows us to give a much more secure scientific ground to object-oriented programming, to which we will devote the remainder of the paper. First, this section presents our theory of class inheritance. Let us begin by considering a typical class-inheritance hierarchy for closed figures, based on Meyer (1997, p.528). It is shown as Figure 10.

Current object-oriented programming has no systematic way of explaining such a hierarchy. However, our generative theory of shape explains it with complete rigor and insight, as follows:

The two basic principles of our generative theory are

- maximisation of transfer

- maximisation of recoverability.

Let us begin by using the second principle. This is realised by the Asymmetry Principle, which recovers symmetries from asymmetries. In particular, the Externalisation Principle says that any use of the Asymmetry Principle for external inference must eventually lead

back to an iso-regular group. These principles predict the class-inheritance hierarchy of Figure 10, in the following way.

Figure 10 A typical class-inheritance hierarchy
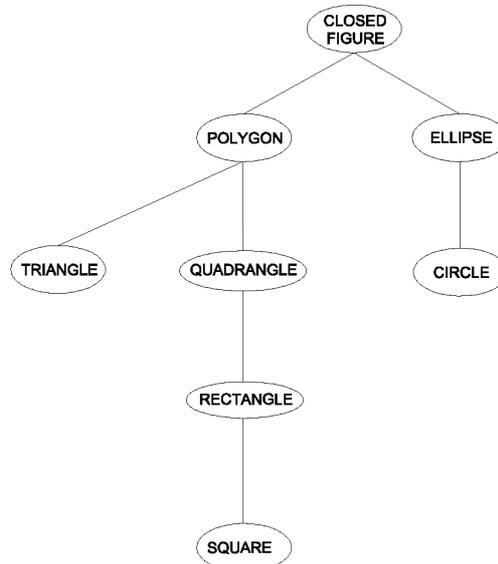
Source: Based on Meyer (1997, p.528)

Observe first that, as one descends through the hierarchy, one is reaching successively more symmetrical states. Thus we see that descendence through the class hierarchy is given by the Asymmetry Principle. Furthermore, this downward use of the Asymmetry Principle is given by external inference, and therefore must be realised by the Externalisation Principle, which implies that the terminal descendant of each branch must correspond to an iso-regular group. These conclusions are summarised as follows.
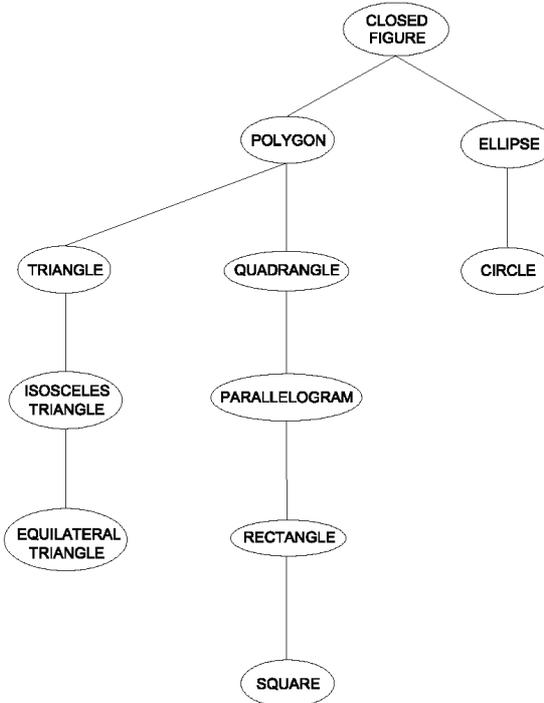
Theory of class inheritance

- *Inheritance is a recovery procedure.*

- *Descendence through the class hierarchy is given by our fundamental rule of recovery: the Asymmetry Principle.*

- *Since the recovery is external at each stage, the Externalisation Principle applies, and therefore, the terminal descendant of each branch corresponds to an iso-regular group.*

We therefore argue that this models, rigorises, and predicts the *discovery* procedure by which software engineers proceed, as follows: By the principles of good software engineering, one first establishes a base class, and subsequently discovers its descendant classes, in a successive manner, thus ensuring that the code of the base class does not have to be rewritten in establishing the code of any of its descendants. What is remarkable is that our theory *predicts* the sequence of descendant classes the software engineer will discover in the development of the software. Thus one will no longer have to wait for the engineer to discover these classes in the usual unguided way.

With this in mind, let us now notice the following: The triangle branch in Figure 10 contains only one TRIANGLE class, and indeed this is not in the form of an iso-regular group. Our theory of inheritance predicts that the software engineer has not actually completed the discovery that will ensue, as the software is developed to become more usable by customers. In particular, our theory predicts that each successive descendant class will remove an asymmetry by external inference. This means that there are two successive classes below the TRIANGLE class: The first is the *isosceles* triangle, and the second and final class is the *equilateral* triangle, which corresponds to the iso-regular group.

Furthermore, notice, also by our theory, that the QUADRANGLE class in Figure 10 is missing below it a crucial class, PARALLELOGRAM, which should be inserted between QUADRANGLE and RECTANGLE. Thus, bringing together the conclusions of this and the preceding paragraph, our principles predict that the full class hierarchy, which the software engineer will eventually discover is that shown in Figure 11.

Figure 11 The rigorous class-inheritance hierarchy predicted by our theory

15 The *is-a* relation

As a result of the above discussion, it is now possible to give a deep algebraic theory of the *is-a* relation that is basic to class-inheritance. As is standardly noted, *is-a* means *sub-class of* rather than *member of*; that is, it really means *is-a-kind-of.* To illustrate the algebraic theory we will develop of this, let us examine the descendant branch starting with the node QUADRANGLE in Figure 11.

One can consider the class text of QUADRANGLE, in the software, as including an invariant stating that there are four sides, and a feature stating that the four side-lengths are real numbers.

Now for the crucial point: Our argument will show that it is fundamentally important to ask what the symmetry group of this structure is. First, we claim that the group is the wreath product:

$$\mathbb{R} \, \text{\textcircled{W}} \, \{e\} = \left[\mathbb{R}_{c_1} \times \mathbb{R}_{c_2} \times \mathbb{R}_{c_3} \times \mathbb{R}_{c_4} \right] \text{\textcircled{S}}_\tau \, \{e\}.$$

There are four copies of the fiber $\mathbb{R}$, but the control group $\{e\}$ is trivial: Its action is to leave each side where it is. This corresponds to the fact that, on an arbitrary quadrangle, there is no symmetry group that carries the sides onto each other, because, typically, the sides have different lengths and the vertices have different angles. The *transfer* structure is therefore trivial, and therefore given by $\{e\}$.

Next, move one step down in the class-inheritance hierarchy (Figure 11) to the next node PARALLELOGRAM. This class inherits the invariant (four sides) and feature (side-lengths are real numbers) from the class above. However, the symmetry group now increases. It is

$$\mathbb{R} \, \text{\textcircled{W}} \, \mathbb{Z}_2$$

where there are again four copies of the fiber, but where the control group as increased to $\mathbb{Z}_2 = \{e, r_{180}\}$, where r_{180} is 180° rotation of the parallelogram (about its centre). This is the only isometry that sends the parallelogram onto itself. Notice that all descendants of the quadrangle will have four copies of the fiber, by inheritance.

Now move one step further down the class hierarchy (Figure 11) to the next node, RECTANGLE. The symmetry group is still larger, thus:

$$\mathbb{R} \, \text{\textcircled{W}} \, [\mathbb{Z}_2 \times \mathbb{Z}_2]$$

where the control group $\mathbb{Z}_2 \times \mathbb{Z}_2$ is the direct product of the reflection group $\mathbb{Z}_2 = \{e, m_V\}$ where m_V is the vertical reflection, and the reflection group $\mathbb{Z}_2 = \{e, m_H\}$ where m_H is the horizontal reflection. Notice that the multiple of reflection m_V with reflection m_H is the rotation r_{180} which, by group closure, must also be in $\mathbb{Z}_2 \times \mathbb{Z}_2$. Therefore, the rotation group $\mathbb{Z}_2 = \{e, r_{180}\}$ of the parallelogram must be a subgroup of the double-reflection group $\mathbb{Z}_2 \times \mathbb{Z}_2$ of the rectangle. In fact, the latter group can be created by what is algebraically called a *group extension* of the former group.

Finally move one step further down the class-inheritance hierarchy (Figure 11) to the bottom node, which is the class SQUARE. Here, the symmetry group is larger still:

$$\mathbb{R} \, \text{\textcircled{W}} \, \left[[\mathbb{Z}_2 \times \mathbb{Z}_2] \, \text{\textcircled{S}}_\tau \mathbb{Z}_2 \right].$$

This extends the group of RECTANGLE by the $\mathbb{Z}_2$ shown on the far right. It is important to notice that the right subsequence $[\mathbb{Z}_2 \times \mathbb{Z}_2] \, \text{\textcircled{S}}_\tau \, \mathbb{Z}_2$ is actually the wreath product $\mathbb{Z}_2 \, \text{\textcircled{W}} \, \mathbb{Z}_2$ discussed in Section 4, and therefore the above group sequence is actually:

$$\mathbb{R} \, \text{\textcircled{W}} \, \mathbb{Z}_2 \, \text{\textcircled{W}} \, \mathbb{Z}_2.$$

This is an iso-regular group. Therefore, by our theory, it terminates the downward branch.

Table 1 Internal symmetry groups of a class inheritance hierarchy

Quadrilateral	$\mathbb{R} \ \widehat{W} \ \{e\}$
Parallelogram	$\mathbb{R} \ \widehat{W} \ \mathbb{Z}_2$
Rectangle	$\mathbb{R} \ \widehat{W} \ [\mathbb{Z}_2 \times \mathbb{Z}_2]$
Square	$\mathbb{R} \ \widehat{W} \ [[\mathbb{Z}_2 \times \mathbb{Z}_2] \ \widehat{S}_\tau \ \mathbb{Z}_2] = \mathbb{R} \ \widehat{W} \ \mathbb{Z}_2 \ \widehat{W} \ \mathbb{Z}_2$

We have therefore demonstrated the following: At each successive class downwards, the symmetry group increases. In fact, the downward hierarchy is given by a sequence of *group extensions*. This information is shown in Table 1.

Definition 3: The symmetry group of all the objects of a class will be called the *internal symmetry group* of that class.

The crucial conclusion we have arrived at is as follows:

Theory of the *is-a* relation: *The sequence of* is-a *relationships down any branch of the class inheritance hierarchy corresponds to a sequence of group extensions of the successive internal symmetry groups of the classes and terminates at an iso-regular group.*

Now, since the internal symmetry group of a class holds for *all* objects of the class, it is an *invariant* of the class. We therefore make a crucial proposal concerning the writing of a class text:

Basic proposal 4: *The internal symmetry group of a class should be written in the invariant clause of the class text.*

The fundamental consequences of this will now be examined.

16 Object-creation

Standardly, what one means, in object-oriented programming, by the fact that an invariant holds for all objects of a class is that it holds in the following two critical run-time situations:

- on object-creation
- before and after the remote call of any routine of the class.

This section examines the first of these, and Sections 17 and 18 examine the second.

In conventional object-oriented programming, the creation procedure of a class produces objects that conform to the invariant clause, in fact, as Meyer (1997, p.466) says, "a creation procedure's formal role is to establish the class invariant". In contrast, in the form of object-oriented programming we are proposing, the relationship is reversed: The class invariant, as internal symmetry group, *provides* the creation procedure. This is accomplished in the following way.

First, we use a basic principle from Leyton (2001):

Symmetry-to-trace conversion principle: *Any symmetry can be redescribed as a trace. The transformations defining the symmetry generate the trace.*

Next, observe that each of the internal symmetry groups is structured as a hierarchy, in which each level is a direct product of isomorphic one-generator groups, and the relation between levels is given by a wreath product. Now we showed in our books (Leyton, 1992, 2001) that any such symmetry group dictates a program, which we call a *canonical plan,* for drawing the figure. It does so in the following way: Each wreath product within the group hierarchy corresponds to a nested do-while loop, where the fiber is a drawing loop, and its control is the drawing loop within which it is nested. The group generator on each level becomes the adder instruction within its corresponding program loop. Note that if the highest level is not transitive in its action on the fiber-group copies immediately below it, then orbits of the fiber-group copies corresponding to the transitive components above are created in parallel. With the correspondences just given between the group products (wreath and direct) and the program structure, the internal symmetry group provides a canonical plan for generating the figure as a trace. Most crucially, we are lead to the following conclusion:

Internal group/creation procedure principle: *The internal symmetry group of a class prescribes, via the canonical-plan realisation of the Symmetry-to-Trace Conversion Principle, the creation procedure creating any object in the class as a trace.*

17 Fundamental structure of a class

In the type of object-oriented programming we are proposing, the invariant clause contains the internal symmetry group of the class. By the theory presented in this paper, the relation between any command operation and the internal symmetry group is one of transfer. We therefore conclude that this gives a profound structuring of the software text:

Fundamental algebraic structure of a class: *Each class is given by a wreath product:*

$$G_{\text{Sym}} \ⓌＷ \ G(C)$$

where G_{Sym} is the internal symmetry group of the class and $G(C)$ is the group of command operations.

18 Class consistency

The new approach to object-oriented programming, that we are proposing, gives a new understanding of class consistency that is far deeper than that which is currently held.

One of the ways we can explain this new understanding is to show that it solves a long-standing problem with respect to the Liskov Substitution Principle (LSP) which states that a routine defined for a base class cannot be violated for any of the latter's derived classes (Liskov, 1988). The LSP is desirable because well-designed code is extendable without modifying already-working code and in the case of class

inheritance, this means that the routines of the base class are maintained after adding the latter's descendant classes. Martin (1996) showed that the LSP is frequently violated in graphical software, with such examples as follows: If one defines, for the RECTANGLE class, a routine that alters the length of a rectangle object relative to its width, then this would violate the invariance conditions of its child class SQUARE, which include having equal sides. Furthermore, as stated by Meyer (1997, p.368), the correctness requirement on an exported routine means that executing the body of the routine – started in any state where the class invariant and precondition both hold – must end in a state in which the invariant and postcondition both hold; i.e., the invariant acts as a consistency condition on the *entire* class of objects. Thus the above rectangle routine would violate the invariance condition of its child class SQUARE, and therefore violate the consistency of that class.

We shall now show that this is solved using our approach. The argument takes a number of steps.

First, according to our Internal Group/Creation Procedure Principle (Section 16), if one created a rectangle directly from the RECTANGLE class, one would use the internal symmetry group of the class, which Table 1 gives as

$$\mathbb{R} \, \mathbb{W} \, [\mathbb{Z}_2 \times \mathbb{Z}_2]$$

and the rectangle would be created purely as a trace from the group in the following way: In the above group sequence, the wreath-product sign and the direct-product sign indicate that the four sides are drawn in two reflectional pairs, which implies that adjacent sides are of independent length.

Next suppose that, instead of creating the rectangle directly from the RECTANGLE class, it were generated in the following alternative way that can often occur in a run-time CAD session: First, at some stage in the session, one has created an object from the class SQUARE. Notice therefore that, by our Internal Group/Creation Procedure Principle, the square has been created purely as a trace using the internal group of its class. This group is different from that of the RECTANGLE class; that is, by Table 1, the group is:

$$\mathbb{R} \, \mathbb{W} \big[[\mathbb{Z}_2 \times \mathbb{Z}_2] \, \mathbb{S}_\tau \, \mathbb{Z}_2 \big] = \mathbb{R} \, \mathbb{W} \, \mathbb{Z}_2 \, \mathbb{W} \, \mathbb{Z}_2$$

which means that the canonical plan draws a side of only a single length, and *hierarchical transfer* does all the rest.

Then suppose that, later in the CAD session, one has needed to apply the stretch operator to this square object, producing a rectangle. It is at this stage that one apparently violates the LSP, because the output of the stretch routine has violated the invariant clause of the SQUARE class, i.e., violated what in standard programs is called the equal-sides rule, or what in our system is much more powerfully given as the internal group $\mathbb{R} \, \mathbb{W} \, \mathbb{Z}_2 \, \mathbb{W} \, \mathbb{Z}_2$.

However, we will now see how our generative theory solves this problem.

Before we begin, it is worth observing that the displayed figure, the rectangle, can have two *perceptual* interpretations to the viewer:

- it could be a *rectangle* in the sense that it has been drawn purely as a trace (i.e., internally)

- according to our psychological research in Figure 9, it can be viewed as a *stretched square*.

These two psychological interpretations therefore correspond to the two interpretations, which the viewer can give as to which software class the figure belongs–the RECTANGLE class or the SQUARE class.

It is the second interpretation that corresponds to the apparent violation of the LSP, i.e., the violation of the invariance clause. However, let us now show how our theory solves this apparent violation, both in the software and the corresponding psychological interpretation. It is solved by using our theory of symmetry-breaking in Section 11. According to that theory, the reason one can interpret the rectangle as a stretched square is that it is being seen as a *transferred* version of a square onto a rectangle. Thus, under this interpretation, the rectangle is given by the following transfer hierarchy:

$$[\; \mathbb{R} \; ⓦ \; \mathbb{Z}_2 \; ⓦ \; \mathbb{Z}_2 \;] \; ⓦ \; \textbf{Stretches}$$

where the square is the bracketed part, i.e., the internal symmetry group, and the stretch component is given by the control group to its right. This means that, in conformance with our theory, the square is not lost in its symmetry breaking, but is transferred onto its symmetry-violating state, the rectangle. Notice that this is embodied in our principle, the Fundamental Algebraic Structure of a Class (Section 17), which says that, in a class, the command group is related to the invariant, the square's internal group, via a wreath product. Thus, by transfer and the algebraic structure, which describes it, the class invariant of the square is not lost. Therefore, under our formulation of object-oriented programming, the LSP is not violated.

It is important to understand that this relates to the fact that our generative theory of shape gives new foundations to geometry that are fundamentally different from Klein's Erlanger program, which is the foundation of 20th century mathematics and physics. As our book (Leyton, 2001) shows in detail: Klein's foundations accord with the conventional view of symmetry-breaking, i.e., the invariants are lost by the groups that break the symmetry actions associated with the invariants. In contrast, by our new foundations, the symmetry-starting states are preserved by the symmetry-breaking actions, owing to the recoverability property of our geometry. As a consequence:

New theory of invariants: *In our generative theory, the invariant is the symmetry ground-state of a generative process. It is an invariant in the sense that all objects of the class possess it by* recoverability. *This is represented algebraically by the fact that the symmetry ground-state is a fiber in the symmetry-breaking wreath products that define the objects of the class.*

To illustrate further, let us continue the example of the transition of the square to the rectangle. Following the above approach, one can create the full sequence given by our psychological results in Figure 9, by the following transfer hierarchy:

$$[\mathbb{R} \; ⓦ \; \mathbb{Z}_2 \; ⓦ \; \mathbb{Z}_2] ⓦ \; \textbf{Stretches} \; ⓦ \; \textbf{Shears} \; ⓦ \; \textbf{Rotations}.$$

This is interpreted as the upward transfer of the square onto the rectangle, which is thereby structured as a stretched square – the latter being then transferred onto the parallelogram, which is thereby structured as a stretched sheared square – the latter being then transferred onto the rotated parallelogram, which is thereby structured as a rotated stretched sheared square.

With this analysis, we are lead to a crucial conclusion. While the downward class inheritance hierarchy is such that an ancestor class does not 'know' anything about the

structure of its descendants, one can, at run-time, reconstruct the hierarchy such that an object on the ancestor level can be reinterpreted as the successive upward transfer of descendants. This process works exactly because it is an example of our theory of *recoverability,* the fundamental rule of which is the Asymmetry Principle, which states that the only recoverable operations are symmetry-breaking ones, and the further rule that all external uses of the Asymmetry Principle conform to our Externalisation Principle, which states that external inference leads ultimately to a past state whose internal structure is an iso-regular group. What the above run-time upward-reconstruction of the hierarchy does is to externalise an object at any ancestor level as a sequence of symmetry-breaking transfers upward through the descendants, the lowest of which is the recovered iso-regular group.

As a result, we propose a new object-oriented operator based on this principle.

Externalisation operator: *Given an object on any level, convert it into the sequence of upward symmetry-breaking transfers of the recovered objects from its descendant classes.*

In this operator, one can give the user the option of selecting the descendant level at which the upward generation starts. For example, a parallelogram can be redescribed as a sheared rectangle. However, according to our Externalisation Principle, the fullest use of the above operator is as follows:

Full externalisation operator: *The application of the Externalisation Operator to an object will be said to be full if the selected starting descendant level is that given by the iso-regular group recovered from that object.*

The above operator tells us how to move from the abstraction hierarchy to the corresponding concrete generative hierarchy. The reverse is also possible. Thus, in prototype-based programming (Ungar and Smith, 1987; Dony et al., 1999), which is a concrete generative process without classes and abstraction, the theory gives a principled means of producing classes from concrete generative sequences. The point is that the mathematics is the same, and is built on the same fundamental principles of maximising transfer and recoverability.

References

Allen, G. (2000) 'CAD/CAM data interoperability strategies', in Dankwort, W. and Hoschek, J. (Eds.): *Digital Products*, Stuttgard: B.G. Teubner, pp.7–25.

Capoyleas, V., Chen, X. and Hoffmann, C.M. (1996) 'Generic naming in generative, constraint-based design', *Computer-Aided Design*, Vol. 28, No. 1, pp.17–26.

Dony, C., Malenfant, J. and Cointe, P. (1999) 'Classifying prototype-based languages', in Noble, J., Taivalsaari, A. and Moore, I. (Eds.): *Prototype-Based Programming Languages*, Springer-Verlag, Berlin.

Kirupaharan, N. and Dayawansa, W.P. (2001) 'Theory of reference frames and biological control', *Mathematical and Computer Modeling*, Vol. 33, Nos. 1–3, January–February, pp.193–198.

Leyton, M. (1986a) 'Principles of information structure common to six levels of the human cognitive system', *Information Sciences*, Entire journal issue, Vol. 38, pp.1–120.

Leyton, M. (1986b) 'A theory of information structure I: general principles', *Journal of Mathematical Psychology*, Vol. 30, pp.103–160.

Leyton, M. (1986c) 'A theory of information structure II: a theory of perceptual organization', *Journal of Mathematical Psychology*, Vol. 30, pp.257–305.

Leyton, M. (1988) 'A process-grammar for shape', *Artificial Intelligence*, Vol. 34, pp.213–247.

Leyton, M. (1989) 'Inferring causal-history from shape', *Cognitive Science*, Vol. 13, pp.357–387.

Leyton, M. (1992) *Symmetry, Causality, Mind*, MIT Press, Mass, Cambridge.

Leyton, M. (2001) *A Generative Theory of Shape*, Springer-Verlag, Berlin.

Liskov, B. (1988) 'Data abstraction and hierarchy', *SIGPLAN Notices*, Vol. 23, p.5.

Martin, R.C. (1996) *The Liskov Substitution Principle*, Report available at http://www.objectmentor.com/resources/articles/lsp.pdf.

Meyer, B. (1997) *Object-Oriented Software Construction*, Prentice-Hall, New Jersey.

NIST (2005) Product Engineering Program, Manufacturing Systems Integration Division, http://www.mel.nist.gove/msid/pe.htm.

Pratt, M. (2004) 'Extension of ISO 10303, the STEP standard, for the exchange of procedural shape models', *Proceedings: Shape Modeling International (IEEE)*, Genova, Italy.

Ungar, D. and Smith, R.B. (1987) 'Self: the power of simplicity', *Object-Oriented Programming Systems, Languages and Applications*, ACM Press, Orlando, Florida, pp.227–242.

Vandenbrande, J.H. and Requicha, A.A.G. (1993) 'Spatial reasoning for automatic recognition of machinable feature in solid models', *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 15, No. 12, pp.1–17.

Notes

[1]Note that, since, generally, fiber-set copies are independent sets, the four infinite lines to not intersect but *overlap*.

[2]First, whereas in Capoyleas et al. (1996) information is lost from points that editing has moved out of the explicit geometry, in our system this information is retained, because our group-theoretic approach extends the explicit geometry via the mechanism of the *occupancy group* as described in Section 3. Thus, if part of a feature such as a slot is moved along a face till part of it is off the face, the information of its full structure (e.g., its bounding edges) is retained within the occupancy structure. Second, constraints are defined in a fundamentally different way. Whereas in Capoyleas et al. (1996) constraints are entities to be input into a constraint-solver, in our system constraints are components in the theory of recoverability and related by inference rules for recoverability (see Chapter 14 of *Generative Theory of Shape* for extensive discussion of constraints). Third, whereas in Capoyleas et al. (1996) representation is frame-independent because frames are regarded as incidental adjuncts to the geometry, in our system representation is strongly frame-based because frames are regarded as explicit components of the geometry. This is part of the fact that our system gives new foundations to geometry that fundamentally oppose the Klein Erlanger program. With respect to editing, this means that regeneration will have available the frame information that is crucial for labelling.