Journal of Logical and Algebraic Methods in Programming 127 (2022) 100774

Contents lists available at ScienceDirect

Journal of Logical and Algebraic Methods in Programming

www.elsevier.com/locate/jlamp

A model of systems with modes and mode transitions

Edwin Beggs [a,*], John V. Tucker [b]

[a] Department of Mathematics, Swansea University, SA1 8EN, Wales, UK
[b] Department of Computer Science, Swansea University, SA1 8EN, Wales, UK

ARTICLE INFO

Article history:
Received 19 July 2021
Received in revised form 29 March 2022
Accepted 30 March 2022
Available online 1 April 2022

Keywords:
Modes of operation
Mode transitions
Abstract simplicial complexes
Presheaf of data types
Autonomous cars

ABSTRACT

We propose a method of classifying the operation of a system into finitely many modes. Each mode has its own objectives for the system's behaviour and its own algorithms designed to accomplish its objectives. A central problem is deciding when to transition from one mode to some other mode, a decision that may be contested and involve partial or inconsistent information. We propose some general principles and model mathematically their conception of modes for a system. We derive a family of data types for analysing mode transitions; these are simplicial complexes, both abstract and concretely realised as geometric spaces in euclidean space $\mathbb{R}^n$. In the simplicial complex, a mode is represented by a simplex and each state of a system can be evaluated by mapping it into one or more simplices. This evaluation measures the extent to which different modes are appropriate for the state and can decide on a transition. To illustrate the general model in some detail, we work though a case study of an autonomous racing car.

1. Introduction

Consider a complex system that is designed to have various modes of operation and behaviour. We suppose that each mode is based on a particular set of objectives for the system's operation and will have a particular set of algorithms designed to accomplish these objectives. Thus, modes have a certain independence from one another and the complex system is the combination of its set of modes. A central problem for a system with independent modes is to decide when it is necessary to transition from one mode to some other mode. Each mode is dependent on its internal state and on data it receives from monitoring the behaviour of the system and its environment. Decisions to change modes may need to be made in the presence of partial or competing information.

To illustrate, think of an autonomous car whose basic modes on a race track are: start, drive, corner, stop; these modes may be refined for dry and wet weather conditions, enhancing the drive and corner modes, making six modes. Off the race track the number of modes multiply: new basic modes are needed for turn right, turn left, reverse, and park; and, most important, alert modes are needed for when the presence of traffic signals, nearby cars and pedestrians are detected and need interpretation, leading to emergency modes for exceptional situations. Modes can overlap and be organised into a sort of hierarchy.

In this paper we will explore theoretically the idea of modes of operation and mode transitions. Starting from scratch, with simple examples, we will propose some general principles about modes and create a new algebraic framework for decomposing and recomposing the behaviour of a system using modes. To do this we develop a family of mathematical concepts to model the framework's components, as follows:

* Corresponding author.
E-mail address: e.j.beggs@swansea.ac.uk (E. Beggs).

https://doi.org/10.1016/j.jlamp.2022.100774

1. *Modes*: To model a finite set of modes and mode transitions, we use an algebraic structure called an *abstract simplicial complex*.
2. *Data*: For computations with the data monitoring the behaviour of the system, to each mode we assign its own state space and algebraic data type.
3. *System Architecture*: To model the structure of the whole system, we assemble the mathematical components belonging to the modes using an algebraic structure called a *presheaf*.
4. *Quantification*: To quantify and visualise the modes and the semantic complexities involved in mode transition decisions we use geometric data types in $\mathbb{R}^n$ called *simplicial complexes*.

Given the number of components and stages needed to develop the framework, it may be helpful to look ahead at how these concepts come together.

Technical strategy. A family of modes for a system is modelled by an *abstract simplicial complex* $\mathcal{C}$ in which a mode is represented and indexed by an element $X \in \mathcal{C}$ called an *abstract simplex*. The algebra of the simplices provides a self-contained hierarchical structure for specifying, comparing and refining the decomposition of a system by a family of modes. Simplicial maps between abstract simplicial complexes, which map simplices to simplices, provide a further way to compare, enhance or simplify models of modes.

Furthermore, from an abstract simplicial complex a number of other algebraic and geometric structures can be derived to organise the local-to-global relationship between modes and the system and to quantitatively evaluate and visualise the relevance of a mode to the current behaviour of the system. In outline, this is done as follows:

To each mode, indexed by a simplex $X \in \mathcal{C}$, we associate a package D_X containing state space S_X and algebraic data type $\mathcal{A}_X$, both of which are local to the mode X. The space S_X contains the possible monitoring data needed by the mode X, and the algebraic data type $\mathcal{A}_X$ contains all the operations and tests needed for the algorithms $\mathcal{B}_X$ that control the system when in that mode.

Now, since the system *is* the collection of the modes, we must bring together and integrate the local structures of the modes X with their individual components in D_X. By viewing the abstract simplicial complex as a category, the state S_X and data type components $\mathcal{A}_X$ can be brought together in a presheaf. Thus, our model of the whole the system has the basic mathematical form of a complicated presheaf.

The evaluation of modes, protocols for mode transitions, and their visualisation also uses a key property of the abstract simplicial complex. From any abstract simplicial complex can be derived a *concrete simplicial complex* in real space $\mathbb{R}^n$. This is a geometric data type that is well known in geometric topology: an n-dimensional simplicial complex is essentially a convex hull of $n+1$ distinct points in a vector space [21]. Thus, for an abstract simplicial complex $\mathcal{C}$, we can construct a unique geometric simplicial object $\Delta_{\mathcal{C}}$ in $\mathbb{R}^n$. Note that the geometric and numerical structure $\Delta_{\mathcal{C}}$ *represents the whole system as it is constructed from the modes*.

Now modes are visualised by the simplices of $\Delta_{\mathcal{C}}$, which can help explain a judgement about the data/evidence for contested choices between modes. The behaviour of a system is evaluated by mapping its local states into simplices by means of a computable approximation $\phi_X : S_X \to \Delta_{\mathcal{C}}$ that provides a measure of the extent to which the behaviour is acceptable for several modes. This is a means to create a global image of the system, and decide when and how modes need to be changed.

Although our approach is to start afresh and propose a new analytical framework, the terminology and examples of systems are well known in computer science. To situate our ideas, at various places we will discuss (i) other usages of the term 'modes' (where relevant), (ii) some other approaches to physical systems that combine analogue and digital data, and (iii) how the mode framework may connect with formal methods for correctness.

Structure of the paper. In Section 2, using simple examples, we give an informal introduction to our concept of modes culminating in a set of abstract principles that informally axiomatise a system of modes for a physical system; these principles we analyse in our mathematical modelling of modes and mode transitions. In Section 3, for the convenience of the reader, we give a relatively detailed introduction to the mathematical ideas about abstract and concrete simplicial complexes that we use to formalise modes and mode transitions. Then, in the next three sections, we give a full formal account of our model of modes and mode transitions using the mathematics of Section 3. In Section 7, to illustrate the components of the model, we give a fully formalised case study by applying the framework to autonomous racing cars. In the penultimate Section 8, with the framework now explained, we look at connections with relevant literatures on hybrid systems and formal specification methods. Finally, in Section 9, we discuss the scope of the modelling technique and its further development. The main examples we have in mind are physical systems, but the methods seem to have wider scope and may usefully apply to human-centred systems (such as classifications in security, privacy, trust and other concepts which can be thought of as modes with different restrictions on access).

We thank the referees for their helpful comments on the first draft of this paper.

2. Aperitif: what is a mode?

We begin with some examples to illustrate our initial ideas about modelling modes and mode transitions, and then give a general characterisation.

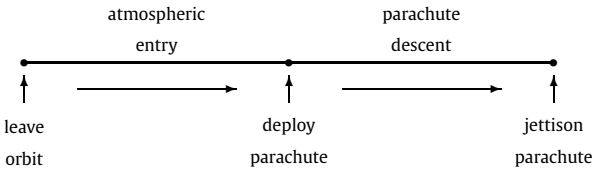


Fig. 1. Visualising a mode transition for a Mars probe.

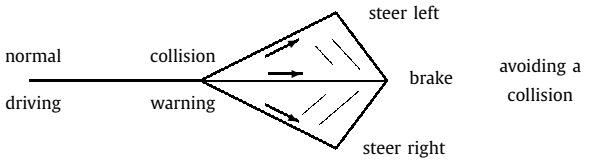


Fig. 2. Visualising a mode transition for a motor vehicle.

2.1. Examples

2.1.1. Control: landing on Mars

A Mars lander goes through the (simplified) five *modes* of: firing engines to leave orbit; atmospheric entry using a heat shield; deploying a parachute; parachute descent; and jettisoning the parachute. As the wreckage of the Mars Polar Lander and the Schiaparelli Lander attest [11,19], the important thing is timing. These modes are different, three of the five are distinct actions of short duration, and two are experiences of significant duration. In Fig. 1, we show the short duration modes as points, and the longer duration modes as lines connecting the points. Thus, we might visualise the changing of the state of the system as a point travelling from one end of the interval to the other. The picture is like a progress bar. The continuous progression contributes to the transparency of the process by reassuring any passengers that the procedure for the descent is on schedule.

Note, a standard picture of the system as a planar graph would look very different – there would be five vertices linked by arrows, and the state of the system would hop discontinuously from vertex to vertex. Such a graph loses the visual representation of the continuous progression from one state to another.

2.1.2. Alert: avoiding collisions

Consider some modes that can be found in driving a car. A driver continually monitors for situations which might cause a problem, such as children playing with a ball by the side of the road. If the situation changes, such as the ball being kicked across the road, then the driver may anticipate a problem and slow down. If a child were to run across the road after the ball, then the driver's priority becomes avoiding a collision. There are obvious options to weigh:

Brake: will the car stop in time?

Steer one way: will the car hit the group at the roadside?

Steer the other way: will the car hit the child or an oncoming car?

Both braking and steering left, or right, are also options.

We can picture the decisions to be made in Fig. 2. As the driver's unease increases, the state of the system moves from normal driving along the edge to collision warning, and the driver may reduce speed. As the child starts running after the ball, we move through collision warning to actively considering the best option for collision avoidance. The states here are represented by two triangles, depending on whether the safest course would be to steer right or left (the driver cannot do both). Suppose that left is indicated and so we are in the upper triangle.

A state evaluated in the triangle means that, given the dangers of taking unnecessary avoiding action, we are merely in a state of readiness and information gathering. If the state is near the vertex *collision warning* then there is little to suggest a mode transition is necessary, whereas a state further away is evidence that some action is required. A state on the edge between *steer left* and *brake* means that avoiding measures will now be taken. The measure or combination of the two actions is given by the position along that edge. Note that the joint mode combining both actions may well be much more complicated than implementing the separate actions.

The different possibilities for the action taken mean that we have a mode for avoiding an object that is naturally 2-dimensional. The planning by an autopilot involves a continuous change of state in the diagram.

Note, the standard planar graph would have 13 vertices, lots of arrows and discontinuous evolution. The idea of continuous evolution means that being nearby physically and in the picture (hopefully) coincide, so the resulting picture is easier to interpret.

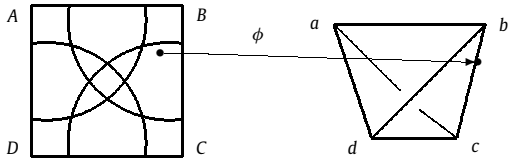


Fig. 3. An assessment map ϕ and a tetrahedron for a system with 4 modes which all intersect.

2.1.3. Intuitions about evaluating states and simplicial complexes

Our two simple illustrations introduce a sense of what we are aiming at: (i) a general notion of mode and mode transition; (ii) a way of evaluating, calibrating or interpreting physical states with respect to different modes; and (iii) a way of continuously visualising the evaluation of the modes geometrically.

The raw data we have available about system behaviour are represented by the states of the system. For a real world physical system, commonly there are physical measurements that are sampled at discrete time points and are approximate.

What we can and need to know about the system is specific to the modes. Each mode will have its own measurements and state space; thus, mode transition requires a means of relating the data available to different modes.

The geometric object is a simplicial complex and is made of n-dimensional components, called n-simplices for integer $n \geq 0$. It is a higher dimensional analogue of a planar graph. For a graph, a 0-simplex is a vertex, and a 1-simplex is a line between two 0-simplices. In general, we then have 2-simplices (filled in triangles) bounded by three 1-simplices (edges) which form the boundary of the triangle. Next we have 3-simplices (filled in tetrahedra) bounded by four 2-simplices which form the faces of the tetrahedron, etc.

From the illustrations, we have a space of measurements, and an *evaluation map* ϕ that classifies or calibrates the states by mapping into a simplicial complex modelling the modes. In Fig. 3 we have supposed that the space of all measurements is a square, and that the regions associated with the modes are quarter disks centred on the four vertices A, B, C, D. The highlighted point in the square is in both the regions centred on B and C, and is associated with modes b and c (but not to modes a or d); it is mapped to a point on the line between b and c.

2.2. Principles for modes

What are the ideas that characterise a system made of modes?

Completeness. A set of modes for a system is a classification of the operation or behaviour of the system. At any time, a system can be in at least one, or more, modes.

Localisation. Each mode for a system chooses and collects its own data to monitor its behaviour and environment. This monitoring data determines the mode's state space and data type for computations.

Combination. When a system is in a number of modes then that situation itself constitutes a mode of the system.

Component. A set of modes for a system consists of a (i) a set of *basic modes* and (ii) *joint modes* made by combining other modes made from basic modes.

These are the most important properties of our concept of modes: if a system is transitioning from basic mode α to basic mode β then there is an intermediate or joint mode α, β which represents the situation that, although the system is in mode α, it is aware that it will need to change to mode β; informally, in symbols, at least three modes are involved:

$$\{\alpha\} \to \{\alpha, \beta\} \to \{\beta\}.$$

The transition could be more complicated. In transitioning from basic mode α to basic mode β, one can imagine that α has to make a choice from a number of modes $\gamma_1, \ldots \gamma_k$, as well as β; informally, in symbols, four modes are involved:

$$\{\alpha\} \to \{\alpha, \gamma_1, \ldots \gamma_k, \beta\} \to \{\alpha, \beta\} \to \{\beta\}.$$

This principle of closure under superposition or overlapping of modes is the key principle that leads us to simplicial complexes.

Two fundamental questions arise in deciding

(i) if and when a system should change from one mode to some other mode, and
(ii) which new mode should be chosen.

As one mode seems less fit for purpose so other modes may become more relevant. However, as data are inexact or incomplete or faulty, the situations are not always easy to recognise. The change from one mode to some other mode is then an inexact process involving decisions by people or algorithms.

Quantification. If a state of the system is meaningful for a number of modes then the relevance or suitability of these modes must be quantified and evaluated.

Thresholds. The transition out of one mode into one other is governed by the results of the quantification and calibration. The decision to move to a new mode may be specified by numerical thresholds.

Typically, thresholds would be used to define an alert and to trigger an intervention (i.e., the change of mode). To these may be added postulates about problem situations that require modes to handle exceptional behaviour.

Visualisations. The quantification of mode states and use of thresholds need to be visualised to suggest instrumentation for modes and transitions.

2.3. Terminology of modes

The term *mode* can appear almost anywhere from fashion and music to logic and statistics – the Oxford English Dictionary currently lists 13 general meanings shaped by their context, one of which is: "Any of a number of distinct ways of operating a device or system". In computer science, the term often plays a role in describing a system by dividing up its operation and behaviour into semantical contexts that are independent to some degree, but must come together to specify the system. We adopt the term mode for this purpose, of course.

Modes and states are widely used terms in computer science and do have several informal or implicit definitions. Commonly they lack a consensus even in particular areas, such as human-computer interaction etc. In software development, the terms appear with different meanings in software design tools – even though there is a standard (ISO/IEC/IEEE 24765).

An interesting and useful study of the terminology as it is used in software tools is Baduel, Bruel, Ober and Doba [4], which cross-references a number of interpretations of modes and states and draws out some commonalities. Our formulation, which started life in [8], is consistent with their recommended attributes.

3. Abstract and geometric simplicial complexes

In the next section, the informal ideas and principles of the last section will be modelled using a series of mathematical concepts that we will introduce in this section. We begin with a very simple example that will help us motivate and explain the purpose of the mathematical concepts.

Example 1. For a car consider a set S which consists of all the data about the car (the position, velocity, tyre pressure, engine temperature etc.). We call S the state space of the system. There are certain concerns, which we have termed *modes*, which may be suggested by this data. For example, certain combinations of data will suggest a failure of the engine cooling system or an imminent collision. For notation, if we set $\mathcal{M}$ to be the set of all such modes then

$$\gamma = \text{“failure of the engine cooling system”}$$

is an element of $\mathcal{M}$ and will correspond to a subset $U_\gamma \subset S$ of states of the system containing the data that flags up this possibility. With the principles of Section 2.2, in mind, note that:

1) We will suppose that every element of S belongs to a subset, even if it is only U_o for $o =$“no concerns”. Thus, the sets in the collection

$$\{U_\alpha : \alpha \in \mathcal{M}\}$$

contain all the states of S.
2) Obviously, some data will correspond to more than one mode, e.g., both an overheating engine and a potential collision, so there will be some non-empty intersections $U_\alpha \cap U_\beta$.
3) We need to be able to differentiate the degree of concerns, e.g., asking a garage to check out the temperature by next week, or immediately, as the engine is so hot that it may fail.
4) These subsets U_α are drawn up by engineers with an idealised global state space S in mind. In reality, the complexity of the car means that different local sensor input and objectives are used to estimate where it is in S and how to respond.

Let S be a set and take a finite collection of subsets $U_\alpha \subset S$ with indices $\alpha \in \mathcal{M}$ such that the union of all the subsets $\cup_{\alpha \in X} U_\alpha = S$; this is a *finite cover* of S and follows the pattern set in Example 1. Historically, this idea has been much studied by algebraic topologists and leads to the idea of a simplicial complex [21], and more recently simplicial complexes have been used in the study of complex systems (see [25] which is also a good introduction to simplicial complexes). Shortly, we will use α etc. to index and name modes, and U_α to collect the states available to mode α.

3.1. Covers and abstract simplicial complexes

Definition 2. An *abstract simplicial complex* $(\mathcal{M}, \mathcal{C})$ is a collection $\mathcal{C}$ of finite subsets of a set $\mathcal{M}$ such that if $Y \subset X$ and $X \in \mathcal{C}$ then $Y \in \mathcal{C}$.

Definition 3. From a cover $U_\alpha \subset S$ for $\alpha \in \mathcal{M}$ of the set S we form an abstract simplicial complex, called the nerve of the cover, by

$$\mathcal{C} = \left\{ X \subset \mathcal{M} : \cap_{\alpha \in X} U_\alpha \neq \emptyset \right\}.$$

Thus $\{\alpha, \beta, \gamma\} \subset \mathcal{M}$ is in $\mathcal{C}$ precisely when $U_\alpha \cap U_\beta \cap U_\gamma$ is not empty. To see that $\mathcal{C}$ is an abstract simplicial complex we note that if $\{\alpha, \beta, \gamma\}$ is in $\mathcal{C}$ then we also must have $\{\alpha, \beta\}$ in $\mathcal{C}$ because if $U_\alpha \cap U_\beta \cap U_\gamma$ is not empty then also $U_\alpha \cap U_\beta$ is not empty.

Definition 4. A map of abstract simplicial complexes $\Psi : (\mathcal{M}, \mathcal{C}) \to (\mathcal{M}', \mathcal{C}')$ is a function $\Psi : \mathcal{M} \to \mathcal{M}'$ so that on subsets if $X \in \mathcal{C}$ then $\Psi X \in \mathcal{C}'$.

We can use maps of abstract simplicial complexes to implement hierarchies for information hiding or classification. To illustrate, we define a refinement of a cover and then continue the story of Example 1.

Definition 5. A cover $\{W_k : k \in \mathcal{M}'\}$ of S is a *refinement* of the cover $\{U_\alpha : \alpha \in \mathcal{M}\}$ if there is a map $\Psi : \mathcal{M}' \to \mathcal{M}$ so that $W_k \subset U_{\Psi(k)}$. It then follows that Ψ is a map of abstract simplicial complexes from the nerve of the cover $\{W_k : k \in \mathcal{M}'\}$ to the nerve of $\{U_\alpha : \alpha \in \mathcal{M}\}$.

Example 6. To make maintenance of the car easier, we split the previous subset U_γ corresponding to $\gamma =$"failure of the engine cooling system" into two (possibly overlapping) subsets for

$$\gamma_F = \text{"fan belt failed" and } \gamma_L = \text{"coolant leak"}.$$

Let $\mathcal{M}'$ denote $\mathcal{M}$ with γ removed and with γ_F, γ_L added.

As the failure of the fan belt and a coolant leak are both failures of the cooling system, the corresponding subsets U_{γ_F} and U_{γ_L} are both subsets of U_γ. In supposing that U_α for $\alpha \in \mathcal{M}'$ is still a cover we have taken the liberty of assuming that all cooling system failures are of these two types – we should really have included an 'other' category. Then the map $\Psi : \mathcal{M}' \to \mathcal{M}$, given by $\psi(\gamma_F) = \gamma$, $\psi(\gamma_L) = \gamma$ and $\psi(\alpha) = \alpha$ for all other cases, shows that we have a refinement of the original cover, and a map of abstract simplicial complexes.

For drivers who do not do their own repairs the new simplicial complex is unnecessary, as they do not care why the engine is overheating. However for a garage doing the repairs the extra information is helpful. From a software point of view, the map Ψ can be used by letting the new modes γ_F, γ_L inherit structures from γ.

3.2. Realisation of abstract simplicial complexes in $\mathbb{R}^n$

The familiar xy plane for 2-dimensional geometry is called $\mathbb{R}^2$ as it uses two copies of the real numbers $\mathbb{R}$. Its elements are ordered pairs (x, y) for x and y real numbers. We have a basis $e_1 = (1, 0)$ and $e_2 = (0, 1)$ and can write any point in the plane as $(x, y) = xe_1 + ye_2$. Similarly for 3-dimensional space $\mathbb{R}^3$ we have points (x, y, z), and if we use new basis elements $e_1 = (1, 0, 0)$, $e_2 = (0, 1, 0)$ and $e_3 = (0, 0, 1)$ we can write $(x, y, z) = xe_1 + ye_2 + ze_3$.

In (for example) $\mathbb{R}^3$ we have points which we call 0-simplices, e.g., e_1 and e_2. The 1-simplices are lines connecting the 0-simplices, e.g., $xe_1 + ye_2$ for real $x, y \geq 0$ with $x + y = 1$. The 2-simplices are triangles spanned by three vertices, e.g., $xe_1 + ye_2 + ze_3$ for real $x, y, z \geq 0$ with $x + y + z = 1$. We extend this to be 3-simplices being tetrahedra, etc. A 3-simplex is visualised in Fig. 3 where the vertices are labelled a, b, c, d.

An n-simplex will have *faces* which are simplices bounded by subsets of its vertices. Thus a 3-simplex will have one 3-face (itself), four 2-faces which are 2-simplices, six 1-faces which are 1-simplices and four 0-faces which are 0-simplices (vertices). Thus in Fig. 3 we have a 1-face labelled by $\{a, c\}$ and a 2-face labelled by $\{a, c, d\}$.

We can use this to define a higher dimensional analogue of a planar graph called a simplicial complex – a graph is an example of a simplicial complex which only contains 0-simplices and 1-simplices:

Definition 7. A *simplicial complex* is a collection of simplices in some space so that the face of any simplex in the collection is also in the collection, and the intersection of any two simplices is a face of both of them.

Fig. 4 gives an example of a simplicial complex which is a union of a 2-simplex and a 1-simplex. These simplices intersect at the common 0-face e_γ.

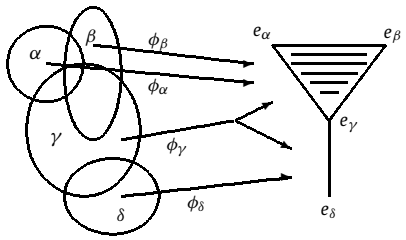


Fig. 4. A partition of unity as a refinement of a cover by four sets.

There is a construction which will give a simplicial complex for every abstract simplicial complex in a functorial manner. In general, this construction uses very high dimensional spaces, which is usually not necessary, but it simplifies the theory. Just as many (but not all) graphs can be drawn in 2-dimensional space we can often draw a simplicial complex in a dimension much smaller than that used in Proposition 8. The large space $\mathbb{R}^{\mathcal{M}}$ is a vector space with basis e_α for all $\alpha \in \mathcal{M}$.

Proposition 8. *To every abstract simplicial complex $(\mathcal{M}, \mathcal{C})$ (as in Definition 2) is associated its standard realisation $\Delta_{\mathcal{C}} \subset \mathbb{R}^{\mathcal{M}}$, as a simplicial complex. The simplex spanned by $X \in \mathcal{C}$ is*

$$\Delta_X = \Big\{ \sum_{\alpha \in X} \lambda_\alpha \, e_\alpha : \lambda_\alpha \in [0,1], \ \sum_{\alpha \in X} \lambda_\alpha = 1 \Big\}.$$

Further, a map of abstract simplicial complexes $\Psi : (\mathcal{M}, \mathcal{C}) \to (\mathcal{M}', \mathcal{C}')$ can be extended to a map of their realisations as $\Delta_\Psi : \Delta_{\mathcal{C}} \to \Delta_{\mathcal{C}'}$ by defining

$$\Delta_\Psi \Big(\sum_{\alpha \in X} \lambda_\alpha \, e_\alpha \Big) = \sum_{\alpha \in X} \lambda_\alpha \, e_{\Psi(\alpha)}.$$

Proof. The simplex Δ_X is a $(|X| - 1)$-simplex where $|X|$ is the size of X, and if $Y \subset X$ then Δ_Y is a face of Δ_X. Then $\Delta_X \cap \Delta_Z = \Delta_{X \cap Z}$ and Definition 7 is seen to be satisfied. □

3.3. Covers and partitions of unity

Example 9. Following from Examples 1 and 6, we might take a state in S which is in U_γ, so we are concerned with the engine overheating. But how concerned should we be? Might it just be a hot day and there is no fault with the car? Is it a critical problem now or a matter for a service in 6 months? Even if it is a critical problem, is there another even bigger problem, such as an imminent collision? We need to quantify the modes to prioritise the response. Further, we would like to quantify this in a manner which humans can understand. This should not be a visualisation constructed after the design of the system, it should be a visualisation which is used both to design the system and to understand its behaviour in real time.

Having created the geometric realisation of the simplex we are now able to use the categorisation of the data in the state space S according to a cover $\{U_\alpha : \alpha \in \mathcal{M}\}$ and its associated abstract simplicial complex $\mathcal{C}$, to create a geometric visualisation in $\Delta_{\mathcal{C}}$. To do this we need an appropriate map from S to $\mathcal{C}$ that contains the information $\phi_\alpha(s)$ which tells us how much we need to be concerned about α when in state $s \in S$ on a scale from 0 to 1.

Definition 10. A *partition of unity* for the cover $\{U_\alpha \subset S \,|\, \alpha \in \mathcal{M}\}$ is a function $\phi_\alpha : S \to [0,1]$ for every $\alpha \in \mathcal{M}$ such that

(1) if $\phi_\alpha(s) \neq 0$ then $s \in U_\alpha$;
(2) $\sum_{\alpha \in \mathcal{M}} \phi_\alpha(s) = 1$ for all $s \in S$.

We then have a function $\phi : S \to \Delta_{\mathcal{C}}$ given by

$$\phi(s) = \sum_{\alpha \in \mathcal{M}} \phi_\alpha(s) \, e_\alpha.$$

Fig. 4 visualises a simplicial complex and partition of unity for a cover by four sets. Note that the triangle in Fig. 4 is shaded to form a 2-simplex precisely because $U_\alpha \cap U_\beta \cap U_\gamma$ is not empty. In specific circumstances we can impose extra conditions on ϕ, e.g., continuity or computability.

3.4. Product simplicial complexes and twofold covers

Example 11. Often systems control independent subsystems. In our car example the passenger comfort subsystem (heating, radio, seatbelt check etc.) might be entirely independent of the engine control subsystem, so one does not need to know anything about the other. This information hiding vastly simplifies the system and so makes it more reliable. In terms of our previous geometry, this can be expressed by a product simplicial complex. However, this is not necessarily the whole story, and another example will illustrate this.

Consider a system controlling autonomous vehicles (trucks and diggers) in a quarry. The operations of the vehicles are largely independent, and coding them as such simplifies the system. However, sometimes the vehicles need common resources (e.g., a digger fills a truck, a truck waits for another truck to unload), and in these cases the vehicles no longer behave independently.

For two subsystems which are mostly independent we can take a product and then look separately at those pairs of modes where there is interaction. These pairs of modes can have their code altered, or we can even split them into further modes using a refinement as in Example 6 and have a simplicial map from a larger simplex into the product. We only make alterations to the independent cases locally, where it is required by the behaviour of the system. This is the method we apply to two cars and a chicane in Section 7.5.

Given abstract simplicial complexes $\mathcal{C} \subset P(\mathcal{M})$ and $\mathcal{D} \subset P(\mathcal{N})$, we have a product complex $\mathcal{C} \times \mathcal{D}$ with vertices $(c, d) \in \mathcal{M} \times \mathcal{N}$ and with simplices, for $k \geq 1$

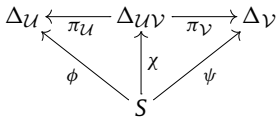
$$\big\{\{(c_1, d_1), \ldots, (c_k, d_k)\} \in P(\mathcal{M} \times \mathcal{N}) : \{c_1, \ldots, c_k\} \in \mathcal{C} \;\&\; \{d_1, \ldots, d_k\} \in \mathcal{D}\big\}. \tag{1}$$

There are maps of abstract simplicial complexes $\pi_1 : \mathcal{C} \times \mathcal{D} \to \mathcal{C}$ and $\pi_2 : \mathcal{C} \times \mathcal{D} \to \mathcal{D}$ given by

$$\pi_1\{(c_1, d_1), \ldots, (c_k, d_k)\} = \{c_1, \ldots, c_k\}$$
$$\pi_2\{(c_1, d_1), \ldots, (c_k, d_k)\} = \{d_1, \ldots, d_k\} .$$

Given two covers U_α (for $\alpha \in \mathcal{M}$) and V_i (for $i \in \mathcal{N}$) of S, we can take the intersection cover $U_\alpha \cap V_i$, and the nerve of the intersection cover $\mathcal{UV}$ is a subcomplex of the product of the nerves $\mathcal{U}$ and $\mathcal{V}$ of the individual covers, and maps to each of the nerves of the original covers. We call this nerve of the intersection complex the nerve of the twofold cover, and this can be generalised to n-fold covers.

Proposition 12. *Given a partition of unity $\phi_{\alpha,i}$ for the cover $U_\alpha \cap V_i$ and elements $e_{\alpha,i}$ of a vector space, we have a map of realisations*



where the maps $\pi_{\mathcal{U}}$ and $\pi_{\mathcal{V}}$ and the single cover partitions of unity are given by

$$\pi_{\mathcal{U}}\Big(\sum_{\alpha,i} \lambda_{\alpha,i}\, e_{\alpha,i}\Big) = \sum_{\alpha} \Big(\sum_{i} \lambda_{\alpha,i}\Big) e_\alpha \;, \quad \pi_{\mathcal{V}}\Big(\sum_{\alpha,i} \lambda_{\alpha,i}\, e_{\alpha,i}\Big) = \sum_{i} \Big(\sum_{\alpha} \lambda_{\alpha,i}\Big) e_i$$
$$\phi_\alpha = \sum_i \chi_{\alpha,i} \;, \quad \psi_i = \sum_\alpha \chi_{\alpha,i} \;.$$

An obvious choice of the twofold partition of unity $\chi_{\alpha,i}$ in terms of partitions ϕ_α and ψ_i for the single covers would be the product $\chi_{\alpha i} = \phi_\alpha\, \psi_i$.

Proof. Given that the $\chi_{\alpha,i}$ are a partition of unity for the cover $U_\alpha \cap V_i$ it is automatic that the ϕ_α and ψ_i in the statement are positive, and they trivially sum to 1. If $\phi_\alpha(x) > 0$ then we must have $\chi_{\alpha,i}(x) > 0$ for some i, and as $U_\alpha \cap V_i \subset U_\alpha$ we see that $x \in U_\alpha$. Similarly for the ψ_i. Finally, note that for the special case of the product partition of unity, if $\phi_\alpha(x)\psi_i(x) > 0$ we need both $x \in U_\alpha$ and $x \in V_i$, and that the sum over α, i is 1. □

Note that our description defines the categorical product, there is a smaller product complex which requires choosing an order $\leq$ on both $\mathcal{M}$ and $\mathcal{N}$, and then insisting that the c_i and d_i in (1) are in increasing order. We do not use this since the indices in the pairings $U_\alpha \cap V_i$ have a real meaning and no sensible order, as discussed in Section 4.

3.5. Categories, functors and presheaves

Definition 13. A *category* consists of a

(1) collection of *objects* $X, Y, V, W, \ldots$.
(2) specification of a set $\mathrm{Mor}(X, Y)$ of *morphisms* for any objects X, Y.
(3) associative composition operation $\circ : \mathrm{Mor}(Y, Z) \times \mathrm{Mor}(X, Y) \to \mathrm{Mor}(X, Z)$.
(4) Every set $\mathrm{Mor}(X, X)$ contains an identity element id_X such that $\theta \circ \mathrm{id}_X = \theta$ and $\mathrm{id}_X \circ \theta = \theta$ for any morphism θ for which $\circ$ is defined.

Proposition 14. *From an abstract simplicial complex $(\mathcal{M}, \mathcal{C})$ we can form a category (which we just call $\mathcal{C}$) which has objects $X \in \mathcal{C}$. For morphisms, $\mathrm{Mor}(Y, X)$ consists of either one morphism, the arrow $Y \to X$, if $Y \subset X$, or no morphisms if $Y \not\subset X$.*

Recall that the appropriate idea of a map between categories is a functor.

Definition 15. A *(covariant) functor* $F : \mathcal{C} \to \mathcal{D}$ between categories specifies an object $F(X) \in \mathcal{D}$ for every object $X \in \mathcal{C}$, and a morphism $F(\theta) : F(X) \to F(Y)$ for every morphism $\theta : X \to Y$, such that $F(\mathrm{id}_X) = \mathrm{id}_{F(X)}$ and $F(\theta \circ \psi) = F(\theta) \circ F(\psi)$ for morphisms.

The following definition corresponds to the topological idea of a presheaf.[1]

Definition 16. For a category $\mathcal{D}$, a $\mathcal{D}$-valued *presheaf* on $\mathcal{C}$ is a functor from $\mathcal{C}$ to $\mathcal{D}$.

This means that for every $X \in \mathcal{C}$ we have an object $D_X \in \mathcal{D}$ and that to every $Y \subset X \in \mathcal{C}$ we have a morphism ${}_X\mathrm{inc}_Y : D_Y \to D_X$. Further, for all $Z \subset Y \subset X \in \mathcal{C}$ we have ${}_X\mathrm{inc}_Y \circ {}_Y\mathrm{inc}_Z = {}_X\mathrm{inc}_Z$.

4. Modes and their mathematical model

In Section 2, we described some examples and general characteristics to motivate the idea of modes and their mathematical representation by objects in $\mathbb{R}^n$. In Section 3 we summarised the mathematics of abstract and concrete simplicial complexes. Now we build the complete model of modes and mode transitions.

4.1. Modes and abstract simplicial complexes

Imagine a physical system that has a number of modes of operation or behaviour. In simple terms, the modes are likely determined by distinct

1. *Mathematical models of the system behaviour*, i.e., which data, equations or algorithms best describe the dynamics in this mode.
2. *Intentions and objectives*, i.e., what should the system be doing, what decisions could or should be made in this mode.

To add to our intuitive examples:

Example 17. Objectives of modes can be illustrated by a plane which has modes of operation that determine its response to the controls. Most obviously, it can start, taxi on the ground, take off, climb, cruise, descend, land, and stop. To these 8 basic modes there are refinements for within flight we can have different control regimes, e.g., an aircraft in turbulence or a stall has radically different handling to normal flight. A plane running short of fuel has several objectives to chose from: (a) continue to its destination, (b) divert to a nearer airfield or (c) attempt a crash landing. These decisions may subdivide, e.g., into (b1) nearest airport with full emergency facilities or (b2) any landing strip. Different data and algorithms are needed for these modes.

Imagine that the actual behaviour of the *whole* physical system can be expressed by states and trajectories in an idealised state space S. We can begin to formalise the properties of modes in Section 2.2 by a categorisation of states: we suppose there is a cover of the state space S by sets, and the set of states that 'belong' to a mode is an element of the cover of the state space. The intersections of sets in the cover have a natural interpretation: a state belonging to two or more elements of the cover means that the state can be in any of these modes. So, for example, a state may be subject to processing by the distinct algorithms associated with each set in the cover. The question arises: Which set of algorithms *should* be in control?

Let us express this mathematically: imagine we have a state space S for a system in which there is a time evolution of a state $s(t) \in S$ for time $t \in T$. To identify and specify a set $\mathcal{M}$ of basic modes of behaviour, we localise the behaviour using a

[1] Presheaves arise in topology and are most commonly used in cohomology with the category of abelian groups.

cover of sets U_α with index set $\alpha \in \mathcal{M}$. A state $s \in S$ of the system is in mode α if $s \in U_\alpha$. Now, we take the nerve of this cover:

$$\mathcal{C} = \left\{ X \subset \mathcal{M} : \cap_{\alpha \in X} U_\alpha \neq \emptyset \right\}.$$

The set $\mathcal{C}$ contains all sets of indexes whose associated subsets in S overlap, e.g.,

$$X = \{\alpha, \beta\} \in \mathcal{C} \iff U_\alpha \cap U_\beta \neq \emptyset.$$

Recalling Section 3.2, we can immediately make the euclidean realisation $\Delta_\mathcal{C} \subset \mathbb{R}^\mathcal{M}$ of the simplicial complex $\mathcal{C}$. We take a partition of unity

$$\phi : S \to \Delta_\mathcal{C}.$$

Then, for a given state $s \in S$ of the system we have

$$\phi(s) = \sum_{\alpha \in \mathcal{M}} \phi_\alpha(s)\, e_\alpha,$$

where e_α for $\alpha \in \mathcal{M}$ is the basis of $\mathbb{R}^\mathcal{M}$. If $\phi_\alpha(s) > 0$ we know that $s \in U_\alpha$.

4.2. Local components of modes

The specification of the system is entirely built by bringing together the specifications of its modes. Thus, we have only the components that belong to modes and our job is to construct a computational structure embracing all the data and algorithms belonging to the modes. Let us take stock: we have an abstract simplicial complex $\mathcal{C} \subset P(\mathcal{M})$ to label the modes of the system; each $\alpha \in \mathcal{M}$ is called a 'basic mode' of the system; and a realisation $\Delta_\mathcal{C} \subset \mathbb{R}^\mathcal{M}$ of the abstract simplicial complex.

To each mode X we associate a *package* D_X of *mode components*:

1. For each mode $X \in \mathcal{C}$, a set S_X of states that defines the data available to the system in mode X.
2. For each mode $X \in \mathcal{C}$, an evaluation function $\phi_X : S_X \to \Delta_\mathcal{C}$ that calibrates the state against the modes.
3. For each mode $X \in \mathcal{C}$, an algorithm $\mathcal{B}_X$ based upon a data type $\mathcal{A}_X$.
4. For any modes $\emptyset \neq Y \subset X \in \mathcal{C}$, partially defined functions ${}_X\text{inc}_Y : S_Y \to S_X$ (here inc stands for 'includes') and ${}_Y\text{proj}_X : S_X \to S_Y$ (here proj stands for 'is a projection of') that relate the data available to the modes X and Y.

However, our modelling raises a number of subtle computational points about the interface between our computational system and reality:

Data. The numerical data available to a mode X depend upon measurements of the system and environment, sampled at various times. This means essentially that the numerical data that makes up each S_X are approximations. Thus, since measurements are rational numbers, if S_X contains only measurements then $S_X \subset \mathbb{Q}^k$ for some k depending on the mode.

Computability. The components of a mode and their inter-dependencies are assumed to be computable.

In our computable approximation to the environment we use a set S_X to contain information about the state of the system in mode X (measurements, approximations, predictions,...) and also about our intent (the orders or instructions for the control system).

To discuss the interface it is convenient to imagine an idealised global state space S to guide our thinking. Of course, the point of modes is that we do not have a workable understanding of the entire system but only of certain modes of operations. Hence, S is an idealisation that is *not* part of the model of the system. If the real system is in state $s \in S$ and $\phi(s) \in \Delta_X$ for some $X \in \mathcal{C}$, then $s \in S$ has a description using data $\tilde{s} \in S_X$. Our modelling began with a collection of sets that we hypothesise are a cover of S. In an ideal case, the state space for mode X would be $S_X = \bigcap_{\alpha \in X} U_\alpha \subset S$.

Since we have no *direct* access to a global state space S, we cannot compute $\phi : S \to \Delta_\mathcal{C}$. We do have a local computable approximation local to mode X:

$$\phi_X : S_X \to \Delta_\mathcal{C} \text{ computes } \phi_X(\tilde{s}).$$

In mode X we have an algebra $\mathcal{A}_X$ of operations and sets, including the local state space S_X and the function ϕ_X. Using this algebra we write an algorithm $\mathcal{B}_X$ to control the system for as long as mode X is considered a suitable mode to be in control.

Suppose that the real system follows a path $s(t) \in S$ over time t. Then while in mode X we have a path in data-space $\tilde{s}(t) \in S_X$, which is calculated by the package D_X using various measurements. In time, the path $s(t)$ will likely leave the subset of S corresponding to mode X and enter that of at least one other mode Y. To implement a change of mode, the

algorithm will invoke functions ${}_X\text{inc}_Y$ or ${}_Y\text{proj}_X$ to change the local state space, where it is convenient to separate the cases where we move to a larger ($X \subset Y$) or a smaller ($Y \subset X$) mode. As these functions will be invoked only when a change of mode is being considered according to the data in S_X, in general they are only partial functions from S_X to S_Y.

To summarise the role of key components: in a given mode $X \in \mathcal{C}$, the algorithm $\mathcal{B}_X$ controls the system using the data available in S_X and the functions in the algebra $\mathcal{A}_X$. The function ϕ_X gives information about which mode we are in, and the partial functions ${}_X\text{inc}_Y$ or ${}_Y\text{proj}_X$ implement the change of mode.

Computation in mode X and the package D_X: For each mode $X \in \mathcal{C}$ we have the following package D_X:

1. A *many sorted algebra* $\mathcal{A}_X$ which contains, among other things
   - **Types** including S_X, $\mathcal{C}$, $\Delta_{\mathcal{C}}$, *check* ...
   - **Functions** including $\phi_X : S_X \to \Delta_{\mathcal{C}}$, ...
   - **Relations**
2. Extensions of the many sorted algebra $\mathcal{A}_X$ which take the form of constants and functions, namely
   - **Oracles** – for i/o data for a mode, explained in Section 4.3
   - **Transfers** – to exchange data between modes, explained in Section 5
   - **Thresholds** – to decide a change of modes, explained in Section 6.5
3. An algorithm $\mathcal{B}_X$ using only types, functions, relations, oracles and transfers etc. from the algebra $\mathcal{A}_X$.

A possible idealised form or template for the algorithm $\mathcal{B}_X$ for mode X is this:

```
   declaration state : S_X; state component variables; auxiliary variables for calculations;
checkTransfer : check
   input state
   loop
     request and receive monitoring data from the environment
     update state with new monitoring data
     evaluate φ_X(state)
     compute actions
     send instructions to environment
     compute 'best' mode Y
     if Trigger then checkTransfer := tran(state, Y)
     if checkTransfer = NotOK then exception
   return
```

Clearly, a central task of the algorithm $\mathcal{B}_X$ is to decide, while in mode X, given a state in S_X, should we go to new mode Y or not?

We must now model the mechanisms involved. The oracles that take care of the environment are easy to formulate, which we do next. The semantics of a transfer between modes is more complicated, as are the evaluation of the state with respect to modes and the use of thresholds to trigger a transition. These latter processes occupy Sections 5 and 6.

4.3. Interface with the environment: oracles

"Reality is merely an illusion, albeit a very persistent one" – Albert Einstein (Attributed).

Here we look at the interface between our model and the environment and the procedure by which the algorithm interacts with external reality. Turing used a term *oracle* for an interaction of an algorithm with a *unknown* device, and we shall generalise this idea slightly, so that the oracles may be used both as input and output devices. Communication with the environment is by oracle calls in control algorithm of the mode, and are implemented using i/o constructs compatible with the programming language of the algorithm.

This interface means that formally algorithms are not dependent on the means of gathering monitoring data. We adapt the operations in an algebra to model interactions with the environment.

Definition 18. An *oracle* is a function of the form

$$\mathcal{O} : \text{output} \times \text{environment} \to \text{input} \times \text{environment}.$$

In terms of the many sorted algebra, it appears as a function

$$\mathcal{O} : \text{output} \to \text{input}$$

where output and input are types, and with the caveat that the value of the function is not repeatable: given an input, it may give different values, simply because the state of the environment may have changed.

Thus, the environment itself is invisible to the algebra, only its indirect effects on the values returned by the oracles can be noticed. How do these functions work? Here are two examples. The first simply collects data from the environment for the algorithm and the second issues an instruction to the environment.

Example 19. Consider a racing car on a track of length L. To establish its position $x \in [0, L]$, we take a measurement oracle $\mathcal{O}_x : \{0\} \times \text{environment} \to [0, L] \times \text{environment}$. The $\{0\}$ simply means that the call of the function has a trivial parameter; the function is called without an argument. Given a state of the system $r \in \text{environment}$, we have $\mathcal{O}_x(0, r) = (x(r), r)$, where we take $x(r)$ to be the position of the car in that state of the system. The value of r has not changed. (In fact the measurement may have changed the system slightly, but here we ignore that.) We also assume that the measurement is instantaneous, if the system was subject to delay we should include a time stamp on the measurement, giving an output $(x, t) \in [0, L] \times T$ where T is the set for time and $t \in T$ is when the observation was made.

The second example effects a change in the environment but may or may not return a confirmation message to the algorithm.

Example 20. Consider a mechanical system with an actuator that rotates a disk. An instruction to an actuator has the form $\mathcal{O}_{rot} : \mathbb{Z} \times \text{environment} \to \{0\} \times \text{environment}$. The effect on the environment is $\mathcal{O}_{rot}(n, r) = (0, \text{turn}(r, n))$, as $\text{turn}(r, n)$ takes the system in state r and turns the disk by $n°$. However, suppose that the algorithm does need to know that the task was successfully completed. Then we could specify $\mathcal{O}_{rot} : \mathbb{Z} \times \text{environment} \to \{\text{OK}, \text{NotOK}\} \times \text{environment}$, where returning OK to the algorithm means that the device doing the turning thinks it was successful, and NotOK indicates a problem.

5. Mode transitions: transferring control to another mode

We have considered the role of the computational packages D_X which are indexed by the modes $X \in \mathcal{C}$. Now we consider the morphisms between the components, which are involved in the transfer of control from one mode to another. In this section, we concentrate on the state space S_X and the information available to the mode X. As we move from mode X to mode Y, how does S_X change and relate to S_Y? There are two cases.

5.1. Moving to a superset: $S_Y \to S_X$ for $Y \subset X$

Consider an example in which there is a change from a simpler to a more complicated situation, in that more factors need to be taken into account.

Example 21. We refer back to Section 2.1.2, and use SL for 'steer left' and B for 'brake'. For the modes $\{SL\}$, $\{B\}$ and $\{SL, B\}$ we have the corresponding state spaces, S_{SL}, S_B and $S_{\{SL,B\}}$. Now suppose that the car is in braking mode B, but that sensor data indicate that to avoid a collision it is also necessary to turn left. Now the car must rapidly fill in information about what is to its left, among other things necessary to safely steer left. In terms of our data structure, S_B is copied into $S_{\{SL,B\}}$ – there are no deletions as information relevant to braking is relevant to the joint operation. However $S_{\{SL,B\}}$ will need some placeholders saying 'fill in with sensor observations as soon as possible'.

There are three general ideas that arise in such cases.

1. *Data*. Initially, in moving to a more complicated situation we do not delete data which we already have. This may be done subsequently, but at first sight we do not know how much of our current information is relevant.
2. *Partiality*. Moving from mode α to $\{\alpha, \beta\}$ may not make sense for all states of the system S_α, so the map $S_\alpha \to S_{\{\alpha,\beta\}}$ on the state space is likely to be only a partial map.
3. *Timing*. Two moves in quick succession, say from S_α to $S_{\{\alpha,\beta\}}$ and then to $S_{\{\alpha,\beta,\gamma\}}$ (with no time to gather or calculate more information in between) would be the same as moving directly from S_α to $S_{\{\alpha,\beta,\gamma\}}$.

These ideas are contained in the following mathematical principle:

Principle 22. For all $\emptyset \neq Y \subset X \in \mathcal{C}$ we have an injective partial map ${}_X\text{inc}_Y : S_Y \to S_X$. For all $\emptyset \neq Y \subset X \subset Z \in \mathcal{C}$ we have ${}_Z\text{inc}_X \circ {}_X\text{inc}_Y = {}_Z\text{inc}_Y$ (the functorial property for inc). For completeness, set ${}_X\text{inc}_X : S_X \to S_X$ to be the identity.

Property 2 leads us to consider the category $\mathcal{S}$ of sets and partial maps. We use Definition 16 to formalise property 3 as a presheaf on the simplicial complex $\mathcal{C}$.

Lemma 23. *For the category $\mathcal{S}$ of sets and partial functions, a $\mathcal{S}$-valued simplicial presheaf is a functor (S, inc) from an abstract simplicial complex $\mathcal{C} \subset P(\mathcal{M})$, with morphisms being inclusion $Y \subset X$, to the category $\mathcal{S}$. This means that for every $X \in \mathcal{C}$ we have a set S_X and that to every $Y \subset X \in \mathcal{C}$ we have a partial map ${}_X\text{inc}_Y : S_Y \to S_X$. Further, for all $Z \subset Y \subset X \in \mathcal{C}$ we have ${}_X\text{inc}_Y \circ {}_Y\text{inc}_Z = {}_X\text{inc}_Z$.*

5.2. Moving to a subset: $S_X \to S_Y$ for $Y \subset X$

Consider an example in which there is a change from a more complicated situation to a simpler one, in that fewer factors need to be taken into account.

Example 24. Continuing from Example 21, we can consider a car doing both SL (steer left) and B (brake) to avoid a collision. If the brake is only being touched very lightly, if at all, it would not cause major disruption to move to the single mode SL. However, if the brake is being applied heavily when the system tries to move into the single mode SL the result is likely to be a major panic, and an urgent move back into the joint mode.

Again, there are three general ideas that arise such cases.

1. *Data.* We may delete data which would be relevant only for factors which are no longer considered.
2. *Partiality.* Moving from mode $\{\alpha, \beta\}$ to α only makes sense if the basic mode β is no longer considered relevant, so the map $S_{\{\alpha,\beta\}} \to S_\alpha$ on the state space is likely to be only a partial map.
3. *Timing.* Two moves in quick succession, say from $S_{\{\alpha,\beta,\gamma\}}$ to $S_{\{\alpha,\beta\}}$ and then to S_α (with no time to gather or calculate more information in between) would be the same as moving directly from $S_{\{\alpha,\beta,\gamma\}}$ to S_α.

These ideas are contained in the following mathematical principle:

Principle 25. For all $\emptyset \neq X \subset Y \in \mathcal{C}$ we have a partial map ${}_X\mathrm{proj}_Y : S_Y \to S_X$. For all $\emptyset \neq Z \subset X \subset Y \in \mathcal{C}$ we have ${}_Z\mathrm{proj}_X \circ {}_X\mathrm{proj}_Y = {}_Z\mathrm{proj}_Y$ (the functorial property for proj). For completeness set ${}_X\mathrm{proj}_X : S_X \to S_X$ to be the identity.

5.3. Combining supersets and subsets, and subsets and supersets

The two data properties in 5.1 and 5.2 have an asymmetry where one map is injective (as data must be copied) and the other is not (as deletions can happen). In the case of the timing properties, two moves in quick succession, say from S_α to $S_{\{\alpha,\beta\}}$ and then back to S_α (with no time to gather or calculate more information in between) would be the same as the identity from S_α to S_α (at least on the domain of the partial map $S_\alpha \to S_{\{\alpha,\beta\}}$). We can use these and other observations to motivate the following principle.

Principle 26. For all $\emptyset \neq X \subset Y \in \mathcal{C}$ we have ${}_X\mathrm{proj}_Y \circ {}_Y\mathrm{inc}_X : S_X \to S_X$ being the identity on the domain of ${}_Y\mathrm{inc}_X : S_X \to S_Y$.

According to whether we are moving along the arrows in $\mathcal{C}$ or in the opposite direction we distinguish the two partial functions

$$ {}_Z\mathrm{inc}_X : S_X \to S_Z \text{ for } X \subset Z \quad \text{and} \quad {}_Z\mathrm{proj}_X : S_X \to S_Z \text{ for } Z \subset X. $$

To invoke these while in mode X we shall simply use one 'function'

$$ \mathrm{tran} : S_X \times \mathcal{C} \to \mathrm{Check} $$

in the many sorted algebra $\mathcal{A}_X$, which if successful will transfer control from the mode. The set Check is a set of exceptions which can be thrown if the transfer does not take place. In mode X the call $\mathrm{tran}(s, Z)$ will have the result:

a) If $Z = X$ then no transfer will take place and the function will return the value $OK \in$ Check.
b) If $X \subset Z$ and $X \neq Z$ then (if successful) the algorithm $\mathcal{B}_X$ will terminate and $\mathcal{B}_Z$ will be started with initialisation $state = {}_Z\mathrm{inc}_X(s) \in S_Z$.
c) If $Z \subset X$ and $X \neq Z$ then (if successful) the algorithm $\mathcal{B}_X$ will terminate and $\mathcal{B}_Z$ will be started with initialisation $state = {}_Z\mathrm{proj}_X(s) \in S_Z$.
d) If none of $X \subset Z$, $X = Z$, $Z \subset X$ are true then no transfer will take place and the function will return the value $notOK \in$ Check.

6. Thresholds for invoking a transition

What can trigger a mode transition? To resolve this we look in more detail at simplicial complexes.

6.1. Paths on simplicial complexes

We use the notation for the realisation of a simplicial complex $\Delta_\mathcal{C}$ from Section 3.2. Then $\Delta_\mathcal{C}$ is the union of the simplices Δ_X for $X \in \mathcal{C}$. We define the interior of Δ_X to be

$$\mathrm{int}\Delta_X = \Big\{ \sum_{\alpha\in X} \lambda_\alpha\, e_\alpha : \lambda_\alpha \in (0,1],\ \sum_{\alpha\in X} \lambda_\alpha = 1 \Big\} \tag{2}$$

in the same terms as Proposition 8, the difference being that all $\lambda_\alpha > 0$ in the interior whereas we just have $\lambda_\alpha \geq 0$ in the simplex.

Lemma 27. *For every element $v \in \Delta_{\mathcal{C}}$ there is a unique $X \in \mathcal{C}$ such that $v \in \mathrm{int}\Delta_X$. If we write $v = \sum_{\alpha\in\mathcal{M}} \lambda_\alpha\, e_\alpha$ then that $X = \{\alpha \in \mathcal{M} : \phi_\alpha(s) > 0\}$. Thus $\Delta_{\mathcal{C}}$ is the disjoint union of the interiors $\mathrm{int}\Delta_X$ for $X \in \mathcal{C}$.*

We also define the faces of Δ_X to be Δ_Y for all $Y \subset X$ where Y has one less element than X.

A path or function of time on $\Delta_{\mathcal{C}}$ can be approximated arbitrarily closely by a piecewise linear path $p(t) \in \Delta_{\mathcal{C}}$. This is basically a join the dots operation using straight line segments. This makes sense in a computational system, as we can only check the position for a discrete set of time values.

Proposition 28. *A line segment contained in $\Delta_{\mathcal{C}}$ must lie entirely within the interior $\mathrm{int}\Delta_X$ for a unique $X \in \mathcal{C}$, with the exception of the end points of the line segment which may either be in $\mathrm{int}\Delta_X$ or on a face (or intersection of faces) of Δ_X.*

As a general principle for a state space S and partition of unity $\phi : S \to \Delta_{\mathcal{C}}$, if we are in state $s \in S$ and $\phi(s) \in \mathrm{int}\Delta_X$ then we should be in mode X, as X is the smallest (or simplest) element of $\mathcal{C}$ which has $\phi(s)$ in its simplex. As we shall see, things are not quite this simple, however we can use this to see why we only considered transitions to a subset or superset mode.

Corollary 29. *Suppose that a line segment contained in $\Delta_{\mathcal{C}}$ has its beginning in $\mathrm{int}\Delta_X$ for some $X \in \mathcal{C}$, but its end is not in $X \in \mathcal{C}$. Then we have two possibilities*

1) *Most of the line segment is contained in $\mathrm{int}\Delta_X$ and its end is in a face (or intersection of faces) of Δ_X.*
2) *Most of the line segment is contained in $\mathrm{int}\Delta_Z$ for some $X \subsetneq Z$.*

Corollary 29 gives us our two cases for transferring from mode X. In case (1) we transfer to a subset mode, and in case (2) we transfer to a superset mode. Any more complicated transition would have to be a composition of these two cases. For example we might transfer from $\{\alpha,\beta\}$ to $\{\alpha,\gamma\}$ by first moving to the face $\{\alpha\}$ of $\{\alpha,\beta\}$ and then to the superset $\{\alpha,\gamma\}$ of $\{\alpha\}$.

To summarise, remembering that the distributed algorithms actually have the final say on requesting transfers, we might propose a method depending on testing for zero values.

Example 30. The state space $S = U_\alpha \cup U_\beta \cup U_\gamma$ is illustrated in Fig. 5, so $\mathcal{M} = \{\alpha,\beta,\gamma\}$ and $\mathcal{C}$ consists of all subsets of $\mathcal{M}$ as all the intersections are non-empty. We write an element of the real simplicial complex $\Delta_{\mathcal{C}}$ in Section 3.2 using the basis $\{e_\alpha, e_\beta, e_\gamma\}$ of $\mathbb{R}^{\mathcal{M}}$ as $a\,e_\alpha + b\,e_\beta + c\,e_\gamma$, or in more compact notation $(a,b,c) \in \Delta_{\mathcal{C}}$ for real $a,b,c \geq 0$ with $a+b+c=1$. Fig. 5 illustrates four states s_1, s_2, s_3, s_4 and their images under a partition of unity $\phi : S \to \Delta_{\mathcal{C}}$:

$$\phi(s_1) = (\tfrac{1}{2}, \tfrac{1}{2}, 0), \quad \phi(s_2) = (\tfrac{18}{20}, \tfrac{1}{20}, \tfrac{1}{20}), \quad \phi(s_3) = (\tfrac{2}{5}, \tfrac{2}{5}, \tfrac{1}{5}) \quad \text{and} \quad \phi(s_4) = (1,0,0).$$

We will switch modes based on a simple test as to whether the coordinates are zero or not.

For the first case, suppose that the path $s(t) \in S$ as a function of time t begins at $s(0) = s_1$ and proceeds to $s(1) = s_3$ so that in $\Delta_{\mathcal{C}}$ the image of the path $\phi(s(t))$ is a straight line path from $\phi(s_1)$ to $\phi(s_3)$. Set $\phi(s(t)) = (a(t), b(t), c(t))$, so initially $c(0) = 0$ and eventually $c(1) = \frac{1}{5}$. We begin in mode $\{\alpha,\beta\}$ and, based on our instructions for changing modes, at the first value of time t we check that has $c(t) > 0$ we transfer to mode $\{\alpha,\beta,\gamma\}$.

For the second case, suppose that the path $s(t) \in S$ begins at $s(0) = s_3$ and proceeds to $s(1) = s_1$ so that $\phi(s(t))$ is a straight line path from $\phi(s_3)$ to $\phi(s_1)$ in $\Delta_{\mathcal{C}}$. Now initially $c(0) = \frac{1}{5}$ and eventually $c(1) = 0$. We begin in mode $\{\alpha,\beta,\gamma\}$ and, based on our instructions for changing modes, at the first value of time t we check that has $c(t) = 0$ we transfer to mode $\{\alpha,\beta\}$.

This example reveals a potential flaw in the method, called the Zeno effect and discussed next in Section 6.2.

However, as far as the computable structure discussed in Principles 22 and 25 is concerned, there are these issues we also need to discuss:

(a) What does the functoriality of the transition functions mean for the data?
(b) What are the domains of the partial functions implementing the transitions?
(c) The use of the idealised function $\phi : S \to \Delta_{\mathcal{C}}$ (not in the model) rather than its computable localisations $\phi_X : S \to \Delta_{\mathcal{C}}$ (in the model).

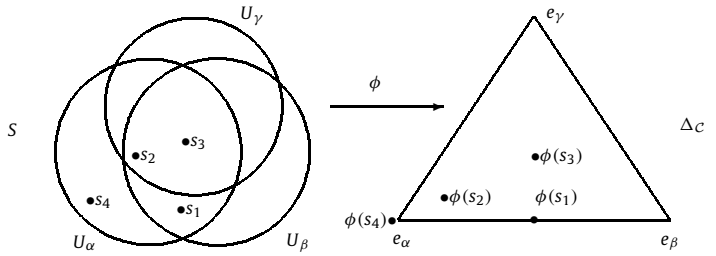


Fig. 5. A partition of unity for Example 30.

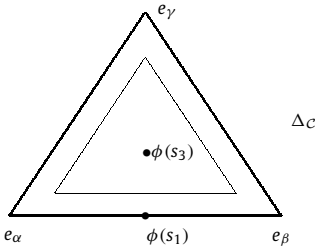


Fig. 6. Avoiding the Zeno effect in Example 31.

6.2. Avoiding Zeno

Zeno of Elea's paradoxes involve infinitely many things happening in finite time. In the theory of hybrid systems, Zeno behaviour refers to a system making an infinite number of discrete changes of state (as opposed to a continuous change) in a finite time [13]. This is something to avoid in changing modes, as the control system would simply fail. In general, it is not possible to guarantee this.

In theory, a perfectly rigid ball bouncing on a perfectly rigid surface will bounce infinitely many times and then come to rest in a finite time. However, as far as any physical measurement is concerned, the ball will effectively come to rest in a finite number of bounces. By imposing a nonzero threshold on the size of bounce that we measure we get around the Zeno behaviour.

Looking at Example 30 as illustrated in Fig. 5, we see that if a path 'bounces' up and down on the horizontal side of the triangle near the point $\phi(s_1)$ then it will change mode very rapidly, from $\{\alpha, \beta, \gamma\}$ to $\{\alpha, \beta\}$ and back again repeatedly. This is because the threshold level for transition one way is the same as that for the other direction. To get round this effect we need to impose a nonzero difference between the threshold levels.

Example 31. Continuing Example 30 as illustrated in Fig. 5, we use different instructions for changing modes. If we begin in the interior of the triangle (2-simplex) $\alpha\beta\gamma$ then we will change mode to a side or a vertex only when we actually touch the boundary of the triangle. However, if we begin from a side or a vertex (a submode of $\{\alpha, \beta, \gamma\}$) we will not change mode to $\{\alpha, \beta, \gamma\}$ unless we cross a boundary strictly inside the interior of the triangle. We visualise this boundary as the inner triangle in Fig. 6. Thus, in the neighbourhood of the points $\phi(s_1)$ and $\phi(s_3)$ in Fig. 6, a transition from $\{\alpha, \beta, \gamma\}$ to $\{\alpha, \beta\}$ will happen on hitting the line given by points $(a, b, 0) \in \Delta_C$, but the transition from $\{\alpha, \beta\}$ to $\{\alpha, \beta, \gamma\}$ will only happen on hitting the inner horizontal line given by points $(a, b, \eta) \in \Delta_C$ for some fixed $\eta > 0$. Now a ball bouncing on the outer horizontal line near $\phi(s_1)$ will only have finitely many bounces higher than the inner horizontal line, so having a threshold value $\eta > 0$ avoids the Zeno effect. (No continuous path can cross from one horizontal line to the other infinitely many times in a finite time.)

6.3. The importance of functoriality

Example 32. We magnify a portion of Fig. 5 (which is explained in Example 30) and display the result in Fig. 7. Then we draw three piecewise linear paths a, b, c from $\phi(s_2)$ to the vertex $\phi(s_4) = e_\alpha$. Suppose that we transfer from mode $\{\alpha, \beta, \gamma\}$ to a subset mode when we touch a side or vertex (imposing other threshold values would require a shift in the paths but not fundamentally alter the consequences).

First start in mode $\{\alpha, \beta, \gamma\}$ at $\phi(s_2)$ and move along path a. The first point on the boundary we hit is $\phi(s_4) = e_\alpha$, so we invoke the transition function ${}_{\{\alpha\}}\mathrm{proj}_{\{\alpha,\beta,\gamma\}} : S_{\{\alpha,\beta,\gamma\}} \to S_{\{\alpha\}}$.

If instead we move along path b then we first move to mode $\{\alpha, \beta\}$ when we get to q and subsequently to mode $\{\alpha\}$, so we get a composition of two transition functions.

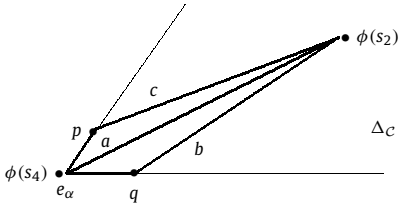


Fig. 7. Small perturbations to paths from Example 32.

If instead we move along path c then we first move to mode $\{\alpha, \gamma\}$ when we get to p and subsequently to mode $\{\alpha\}$, so we get a composition of two transition functions.

Now we can imagine that p and q are so close to the vertex e_α that there is effectively no time for measurement or calculation between them and reaching the vertex, and the three paths a, b, c would be effectively indistinguishable. To avoid the complication of a discontinuity in the data given by a continuous deformation of the paths we would require that for p and q sufficiently close to the vertex that the transition functions gave the same result, i.e.,

$$_{\{\alpha\}}\mathrm{proj}_{\{\alpha,\beta,\gamma\}} = {}_{\{\alpha\}}\mathrm{proj}_{\{\alpha,\gamma\}} \circ {}_{\{\alpha,\gamma\}}\mathrm{proj}_{\{\alpha,\beta,\gamma\}} = {}_{\{\alpha\}}\mathrm{proj}_{\{\alpha,\beta\}} \circ {}_{\{\alpha,\beta\}}\mathrm{proj}_{\{\alpha,\beta,\gamma\}} \ .$$

A more general argument than Example 32 would give the general functorial property for proj in Principle 25, and reversing the paths would give the functorial property for inc in Principle 22. (The functorial property is not strictly necessary, but without it the resulting discontinuity when the path varies would complicate a geometric analysis of the system.)

6.4. The philosophy of the transition functions

It is important to see how transition functions link to information management.

Consider moving from a mode X to a simpler subset mode $Y \subset X$ when we no longer believe that its complement $X \setminus Y$ in X is relevant. In agreement with William of Ockham we establish a principle to move to a simpler (subset) mode where possible.[2]

Conversely, consider moving from mode X to a more complicated superset mode. This is $X \subset Z$ is more difficult as we are moving into an unknown country where there are factors in mode Z that mode X was never designed to understand. Firstly, we would not want to move from mode X if we believe that it is doing well; there has to be a motivation to move – a crisis brewing. However, not so obviously, there is a problem when this crisis is too large. It is not that we cannot change mode in a time of great crisis (indeed there may be little alternative), it is rather that we would be so far into an unknown country that the result of doing so would not be predictable.

6.5. The domain of the transition functions

To apply the principles of Section 6.4, we need to quantify the extent to which a mode or subset of a mode is relevant. We need functions to measure our 'belief' in a mode's fitness for purpose:

$$\mathbb{B}(Y) : S \to [0,1] \quad \text{and} \quad \mathbb{B}_X(Y) : S_X \to [0,1].$$

The function $\mathbb{B}(Y)$ formalises our *belief*, as an element of $[0,1]$, that the subset $Y \subset \mathcal{M}$ is relevant in state $s \in S$. In addition to the abstract picture for $s \in S$ we have $\mathbb{B}_X(Y)$ as our approximation in mode X, using $\phi_X : S_X \to \Delta_C$ and $\tilde{s} \in S_X$. Rather than use a general theory of belief and evidence (e.g., [20]), referring to the notation in Definition 10 we can simply use the sum

$$\mathbb{B}(Y)(s) = \sum_{\alpha \in Y} \phi_\alpha(s) \ , \quad \mathbb{B}_X(Y)(\tilde{s}) = \sum_{\alpha \in Y} \phi_{X\alpha}(\tilde{s}) \ . \tag{3}$$

Consider the two forms of transition.

Moving from a mode X ***to a subset mode*** $Y \subset X$. In Examples 31 and 32 we transferred from mode X to $Y \subset X$ when our path in Δ_X touched the boundary. When we are at a point in the boundary $\psi_X(\tilde{s}) \in \Delta_Y \subset \Delta_X$ then by definition $\mathbb{B}_X(X \setminus Y)(\tilde{s}) = 0$. However, as the initiation of mode transfer is the responsibility of the algorithm $\mathcal{B}_X$ it is best to allow some *wiggle room* by a small parameter $\epsilon_{X \to Y}$. The domain is illustrated in Fig. 8.

[2] Frustra fit per plura quod potest fieri per pauciora = It is futile to do with more things that which can be done with fewer. William of Ockham *Summa Totius Logicae*, circa 1323 A.D.

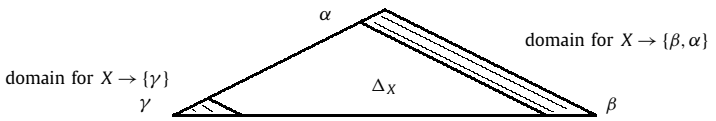


Fig. 8. Illustration of the domains of the transition functions ${}_Y\mathrm{proj}_X : S_X \to S_Y$ for $X = \{\alpha, \beta, \gamma\}$.

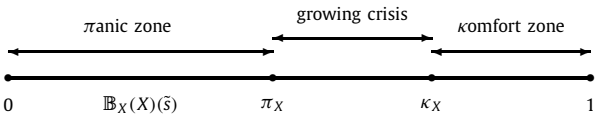


Fig. 9. The comfort and panic zones for a mode X in state $\tilde{s} \in S_X$.

Definition 33. For modes $Y \subset X$ the domain of the transition function ${}_Y\mathrm{proj}_X : S_X \to S_Y$ includes the following set, for a given $\epsilon_{X \to Y} > 0$:

$$\left\{\tilde{s} \in S_X : \mathbb{B}_X(X \setminus Y)(\tilde{s}) < \epsilon_{X \to Y}\right\}.$$

Moving from a mode X ***to a superset mode*** $X \subset Z$. In mode X in state $\tilde{s} \in S_X$ the number $\mathbb{B}_X(X)(\tilde{s})$ measures how confident mode X is that it models state $\tilde{s} \in S_X$ satisfactorily. We highlight two subjective values of belief, the values $0 < \pi_X < \kappa_X < 1$ where κ_X bounds the *κomfort zone* and π_X the *πanic zone* as in Fig. 9.

If we are in the comfort zone for mode X then we are happy to stay in mode X (cf. Ockham's razor: we do not move to a superset unless we have to).

In the 'growing crisis' zone we look for a superset mode to move to.

In the panic zone the search becomes urgent as there is serious doubt about how effective mode X is at controlling the system.

However, for any particular $X \subset Z$ the domain of the transition function ${}_Z\mathrm{inc}_X : S_X \to S_Z$ is rather more difficult, as we are moving into an unknown country where there are factors in mode Z that mode X was never designed to understand.

One obvious condition is that we would like a definite reason to have all the new elements in Z (Ockham's razor again), so we could ask for $\phi_{X\beta}(\tilde{s}) > 0$ for all $\beta \in Z \setminus X$.

Another obvious condition is that we would prefer to jump into the comfort zone of mode Z, as there might be little point in changing from one crisis to another.

Definition 34. For $X \subset Z$ the domain of the transition function ${}_Z\mathrm{inc}_X : S_X \to S_Z$ includes

$$\left\{\tilde{s} \in S_X : \pi_X \leq \mathbb{B}_X(X)(\tilde{s}) \;\&\; \kappa_X \leq \mathbb{B}_X(Z)(\tilde{s}) \;\&\; \phi_{X\beta}(\tilde{s}) > 0 \text{ for all } \beta \in Z \setminus X\right\}$$

To summarise this, we can move out of X if

(i) we are not in the panic zone for X;
(ii) if we can move into the comfort zone for the new mode Z (or at least what X estimates is the comfort zone for Z); and
(iii) if each new element $\beta \in Z$ is justified by $\phi_{X\beta}(\tilde{s}) > 0$.

Note that we have allowed the possibility of moving from a comfort zone to a comfort zone if the algorithm wishes.

6.6. The existence of S and the consistency of the transition functions

To shape our narrative, we have spoken about the idealisations of the global state space S and the partition of unity $\phi : S \to \Delta_{\mathcal{C}}$. However, our theoretical model of a system is based on the idea that to compute we have *only* local sets of states S_X and computable functions $\phi_X : S_X \to \Delta_{\mathcal{C}}$ that belong to the modes. Actually, this is the *raison d'être* of modes: modes address the problem that S may not exist in any meaningful sense. Whilst a simple physical system may have a global S as a workable mathematical abstraction, what is S when we have an autonomous vehicle in a city?[3]

All we have are the local structures S_X, and any meaning for the global system depends upon gluing the local modes together. However, there are consistency issues which arise for the gluing procedure, which in our case leads to criteria on the transition functions.

[3] Just as complex, what could be S in some multiple agency social services situation? cf. 9.3.

Our first stage in gluing the modes together is understanding that the algorithm may no longer be right for the behaviour of the system: if the algorithm is going wrong, then at least it realises that it may be going wrong.

Example 35. A control system in mode X involving a pendulum is written using the formula for a small amplitude oscillation and simple harmonic motion. During the operation of the system, energy is fed into the pendulum and its amplitude of oscillation increases significantly. As this happens the formula will fail to give a good approximation to the motion and the whole control system may collapse. What we need is that the algorithm can flag up that it is running into problems, and our standard method of setting such a flag is through the value of ϕ_X on the state $\tilde{s} \in S_X$.

Principle 36. Following Fig. 9 we assume that

$$\mathcal{W}_X = \left\{\tilde{s} \in S_X : \pi_X \leq \mathbb{B}_X(X)(\tilde{s})\right\}$$

is a set of states which is modelled 'reasonably well' by mode X, where $\pi_X \in (0,1)$ is the πanic level. By modelled 'reasonably well' we assume that for a state $\tilde{s} \in S_X$ a computation carried out on the system by the algorithm for mode X is likely to give a good answer, but no such guarantee exists outside $\mathcal{W}_X$.

One may ask: Why we do not simply restrict the local states to this set in the first place, as then all our calculations would be 'good'. The reason is for many realworld stories where something has gone wrong, the path to safety has been through possibilities rather than certainties. By deleting all but guaranteed options, we merely make it more likely that we will run out of options. Stated alternatively in terms of our hypothetical state space S, if we reduce the size of the sets in a cover we may not get a cover.

The next thing to do is to examine the consistency of the functions ϕ_X for inc (see Definition 34) and proj (see Definition 33). Of course, this compatibility will only be expected to work in a subset of $\mathcal{W}_X$ (see Principle 36) as all our calculations are suspect outside that set. Compatibility corresponds to the following diagram, commuting on the specified domains intersected with $\mathcal{W}_X$

$$\begin{array}{ccc} S_X & \xrightarrow{\phi_X} & \Delta_{\mathcal{C}} \\ {\scriptstyle {}_Z\mathrm{inc}_X / {}_Z\mathrm{proj}_X}\downarrow & \nearrow_{\phi_Z} & \\ S_Z & & \end{array} \tag{4}$$

Finally we come the composition ${}_X\mathrm{proj}_Z \circ {}_Z\mathrm{inc}_X : S_X \to S_X$ for $X \subset Z$ in Principle 26. Definition 34 gives the guaranteed domain of ${}_Z\mathrm{inc}_X : S_X \to S_Z$, and this is a subset of $\mathcal{W}_X$, namely

$$\left\{\tilde{s} \in S_X : \pi_X \leq \mathbb{B}_X(X)(\tilde{s}) \text{ and } \kappa_X \leq \mathbb{B}_X(Z)(\tilde{s}) \text{ and } \phi_X(\tilde{s})_\beta > 0 \text{ for all } \beta \in Z \setminus X\right\}$$

so we deduce that

$$\mathbb{B}_X(Z \setminus X)(\tilde{s}) = \mathbb{B}_X(Z)(\tilde{s}) - \mathbb{B}_X(X)(\tilde{s}) \leq 1 - \pi_X$$

so by the consistency of ϕ_X and ϕ_Z in (4) we have for $\tilde{r} = {}_Z\mathrm{inc}_X(\tilde{s}) \in S_Z$ that $\mathbb{B}_Z(Z \setminus X)(\tilde{r}) \leq 1 - \pi_X$. By Definition 33, the image is contained in the domain of ${}_X\mathrm{proj}_Z : S_Z \to S_X$ if $1 - \pi_X < \epsilon_{Z \to X}$.

Now, in this case the condition that the composition is the identity would make formal sense, and even if the domains and images did not quite match there might be a restriction of the domain on which it made sense. However, similarly to the functoriality discussed in Section 6.3, we should regard this condition as one which makes reasoning about the system easier, and therefore should be a condition which is true as far as possible, rather than a condition where we have to mess with everything just to make sure that it applies exactly.

It may be worth pausing and asking just what is the model of modes mathematical. In topology, a functor from a category, whose objects are given by a cover of a topological space, to another category is called a presheaf (see Definition 16). We assume a fixed concrete cover of S and the functorial conditions in Section 6.3, which may not always be exact, to make a similar idea. Using the compatibility conditions (4) and the one sided identity condition of Principle 26 we try to glue the objects for each element of the cover together as seamlessly as possible, and using a further slight abuse of notation we might call this a sheaf construction. The model of modes is mathematically this:

Definition 37. A *abstract simplicial sheaf datatype* is a simplicial complex $\mathcal{C}$ with a local computable structure as given in Section 4.2 obeying (to a reasonable extent) the functorial conditions in Section 6.3, the one sided identity condition of Principle 26 and the compatibility conditions (4).

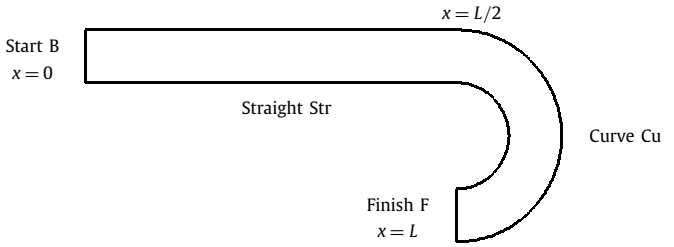


Fig. 10. A track for racing cars.

6.7. A summary in preparation for a case study

In describing the model we have made a distinction in our theorising between the real physical system and its operating environment and the digital system that is designed to approximate and control it.

Real world idealisation. Consider a complex system that must perform various tasks. To have an effective mathematical descriptions and appropriate algorithms for these tasks, the system is designed in pieces, which must be combined to specify the system. These pieces we have called *modes*. The patchwork of modes is formalised by imagining a global state space description S for the whole system and postulating idea of a cover of the state space S. This cover determines all the local state spaces that are the basis of the modes. The key technical idea is that these local state spaces, and the modes that depend upon them, have the structure of an *abstract simplicial complex* $\mathcal{C}$. The modes are represented by simplices. The actual control of the system is achieved by *sensors* and *actuators*. The flow within the system is given by conditions for *transitions* between modes. The most important ingredient of these conditions is the *partition of unity* $\phi : S \to \Delta_{\mathcal{C}}$ which determines the fitness of a mode X to control the system in state $s \in S$.

Computable world idealisation. The programmers construct sets S_X which hold the data for the picture of reality available to the mode X. The control of the system in mode X is implemented by an algorithm $\mathcal{B}_X$ written using a many sorted algebra $\mathcal{A}_X$. The communication with the real world sensors and actuators is performed by oracles, which are formally functions in $\mathcal{A}_X$. The local representation of the partition of unity $\phi_X : S_X \to \Delta_{\mathcal{C}}$ is a function in $\mathcal{A}_X$. Two special components effect the performance of modes.

The confidence levels. To judge the effectiveness of a mode X to control the system, using ϕ_X as shown in Fig. 9 we split into a *comfort zone* (i.e., above κ_X mode X is doing a 'good' job) and an *panic zone* (i.e., below the level π_X mode X may not control the system sufficiently well).

Transitions between modes. Transitions are given by partial functions ${}_Y\mathrm{proj}_X : S_X \to S_Y$ for $Y \subset X$ and ${}_Z\mathrm{inc}_X : S_X \to S_Z$ for $X \subset Z$, and transition is initiated by the call of a formal function in $\mathcal{A}_X$ by the algorithm. Their (minimal) domains are described in Definition 34 and Definition 33. There is a consistency condition between these transition functions and ϕ_X described in (4).

A desirable condition, but not essential, on the transition functions is that they are functorial on a reasonable domain (see Principle 22 and Principle 25), and that we have a one sided inverse property as in Principle 26.

7. Case study: autonomous racing cars

We give a toy example to illustrate the components of the model, it is not meant to be a practical example. To illustrate the construction of the modes and mode transitions we use a track and one car. Then we introduce a second car to illustrate independence and product complexes, with the chicane forcing an interaction between the cars (see Section 3.4).

7.1. Single cars and the presheaf

Consider a racing track, pictured in Fig. 10, of length L with a car that goes from the start line B to the finish line F; the aim is to do this safely and in a short time. We take the transition from straight to curve to be the half way point, $L/2$. The cars have maximum speed 120 km/hr and cannot go backwards.

As this is a paper more about data than control, we choose a very simple control algorithm. We suppose that the speed of the car is given by the setting of a power controller which is marked $[0, 120]$, and that there is no need to monitor the speed independently.

A state space for the whole system is $S = [0, 120] \times [0, L]$, which defines velocity and position. We take an open cover of S consisting of open sets representing the straight and curve part of the track:

$$U_{\{\mathrm{Str}\}} = [0, 120] \times [0, \tfrac{3}{4}L)\ ,\ U_{\{\mathrm{Cu}\}} = [0, 120] \times (\tfrac{1}{4}L, L]\ ,\ U_{\{\mathrm{Str,Cu}\}} = [0, 120] \times (\tfrac{1}{4}L, \tfrac{3}{4}L)\ .$$

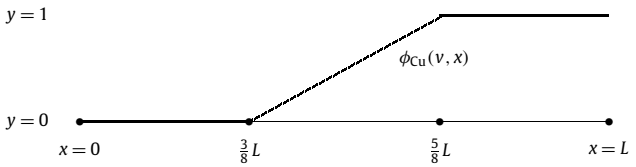


Fig. 11. The function $\phi_{\text{Cu}} : S = [0, 120] \times [0, L] \to [0, 1]$.

The abstract simplicial complex is $\mathcal{C} = \big\{\{\text{Str}\}, \{\text{Cu}\}, \{\text{Str}, \text{Cu}\}\big\}$ and we specify the partition of unity $\phi : S = [0, 120] \times [0, L] \to \Delta_{\mathcal{C}}$ by graphing the ϕ_{Cu} component (the component taking nonzero values only in $U_{\{\text{Cu}\}}$, see Definition 10) in Fig. 11. For this choice of function note that $\phi_{\text{Cu}}(v, x)$ is independent of v and only depends on $x \in [0, L]$. Then we have $\phi_{\text{Str}} = 1 - \phi_{\text{Cu}}$.

7.2. The data types for the modes

As the original state space is already given in simple numerical terms, for the data structures we can simply take the sets themselves, i.e., $S_X = U_X$ above. Recalling 4.2, in this simple case we can simplify the description of the algebras for the three modes by using a common inherited structure. We suppose that $\mathcal{A}_X$ contains the datatypes:

Type	Comment	$X = \{\text{Str}\}$	$X = \{\text{Str}, \text{Cu}\}$	$X = \{\text{Cu}\}$
S_X	local state space	$[0, 120] \times [0, \frac{3}{4}L)$	$[0, 120] \times (\frac{1}{4}L, \frac{3}{4}L)$	$[0, 120] \times (\frac{1}{4}L, L]$
speed_X	speed of car in km/s	$[0, 120]$	$[0, 120]$	$[0, 120]$
position_X	position of car in $[0, L]$	$[0, \frac{3}{4}L)$	$(\frac{1}{4}L, \frac{3}{4}L)$	$(\frac{1}{4}L, L]$
$\mathcal{C}$	$\big\{\{\text{Cu}\}, \{\text{Str}, \text{Cu}\}, \{\text{Str}\}\big\}$			
check	{OK, NotOK}			
null	$\{0\}$			
$\Delta_{\mathcal{C}}$	$[0, 1]$			

In this simple case we avoid special notation for simplicial complexes by identifying $\Delta_{\mathcal{C}}$ with the unit interval $[0, 1]$, where the vertex e_{Str} corresponds to $0 \in [0, 1]$ and e_{Cu} corresponds to $1 \in [0, 1]$. Thus the position of the system in $\Delta_{\mathcal{C}} = [0, 1]$ is simply given by the value of ϕ_{Cu} in Fig. 11. In addition $\mathcal{A}_X$ contains the following functions, oracles and transfers, given in a general case for mode X.

Function	Value	Comment
$\text{updateSpeed} : \text{speed}_X \times S_X \to S_X$	$\big(s_1, (s_2, p)\big) \mapsto (s_1, p)$	update speed
$\text{updatePosition} : \text{position}_X \times S_X \to S_X$	$\big(p_1, (s, p_2)\big) \mapsto (s, p_1)$	update position
$\text{retrieveSpeed} : S_X \to \text{speed}_X$	$(s, p) \mapsto s$	retrieve speed
$\text{retrievePosition} : S_X \to \text{position}_X$	$(s, p) \mapsto p$	retrieve position
$\phi_X : S_X \times \mathcal{M} \to [0, 1]$	$(s, \alpha) \mapsto \phi_X(s)_\alpha$	components of ϕ_X
Oracle		
$\mathcal{O}_{\text{pos}} : \text{null} \to \text{position}_X$	real world position	input position
$\mathcal{O}_{\text{power}} : \text{speed}_X \to \text{check}$	OK if successful notOK if a problem	output power (speed)
Transfer		
$\text{tran} : S_X \times \mathcal{C} \to \text{check}$	OK if successful NotOK if a problem	$\text{tran}(s, Z)$ transfers control to $Z \in \mathcal{C}$ with initial state $s \in S_X$

The function $\phi_X : S_X \to \Delta_{\mathcal{C}} = [0, 1]$ is the restriction of ϕ to S_X. The oracle $\mathcal{O}_{\text{pos}}$ is defined by $\mathcal{O}_{\text{pos}}(0) \in [0, L]$ being the position of the car when the oracle is called (we suppose no delay and no error), and calling the oracle does not affect the *environment* (see the discussion in Example 19). (We should also have an exception for being out of the specified range, but have chosen not to implement this.) As the input value to the function is null there is no information transferred to the sensor making the measurement.

The oracle $\mathcal{O}_{\text{power}}(v)$ is defined by setting the power controller of the car to speed $v \in [0, 120]$ (cf. another actuator or control mechanism in Example 20). In this case the value returned by the power controller is either OK (the operation has been carried out) or NotOK (a problem occurred).

7.3. Mode transitions

Suppose that we are in mode $X \in \mathcal{C}$. Mode transitions to simpler (subset) modes $Y \subset X$ take priority over moving to more complicated (superset) modes. We shall move to a subset mode on leaving the interior of Δ_X (see (2)) in the simplicial complex. The only time we expect to make a transition to a subset mode is from $\{\text{Str}, \text{Cu}\}$ to $\{\text{Cu}\}$, so this appears only in algorithm $\mathcal{B}_{\{\text{Str},\text{Cu}\}}$ in Section 7.4 where we change mode only when $t := \phi_X(state, \text{Cu})$ takes the value 1.

The only time we expect to make a transition to a superset mode is from $\{\text{Str}\}$ to $\{\text{Str}, \text{Cu}\}$. From Section 6.5 in mode $X = \{\text{Str}\}$ the belief that that mode is doing a good job modelling the system is $\phi_{\text{Str}} = 1 - \phi_{\text{Cu}}$. This has critical values: the 'comfort level' $0 < \kappa_X < 1$ where we consider moving to a larger (superset) mode in the subset of S_X, and a 'panic level' $0 < \pi_X < \kappa_X < 1$ where we consider it urgent to move to a superset mode. To make the transition in plenty of time we set a large value of $\kappa_{\{\text{Str}\}}$, say $\kappa_{\{\text{Str}\}} = \frac{9}{10}$. To make sure that we perform the transition before the end of the straight we set $\pi_{\{\text{Str}\}} = \frac{6}{10}$. As a result the algorithm $\mathcal{B}_{\{\text{Str}\}}$ has the test $\phi_X(state, \text{Str}) < \frac{9}{10}$ for mode transfer.

A consequence of the simplicity of this system is that the transition functions themselves are rather boring: all state spaces are just $[0, 120]$ cross a suitable subset of $[0, L]$, and all transition functions are inclusion maps on similar sets.[4] We also set all the calculated partition of unity ϕ_X to be the abstract partition ϕ restricted to S_X. Thus the main interest for the transition functions is checking their domains. From Definition 33 we have

$$\begin{aligned}
&\text{Dom}\big({}_{\{\text{Str}\}}\text{proj}_{\{\text{Str},\text{Cu}\}}\big) \supset \left\{\tilde{s} \in S_{\{\text{Str},\text{Cu}\}} : \phi_{\text{Cu}}(\tilde{s}) < \epsilon_{\{\text{Str},\text{Cu}\}\to\{\text{Str}\}}\right\} \\
&\quad = [0, 120] \times \left(\frac{1}{4}L, \frac{3 + 2\epsilon_{\{\text{Str},\text{Cu}\}\to\{\text{Str}\}}}{8}L\right) \\
&\text{Dom}\big({}_{\{\text{Cu}\}}\text{proj}_{\{\text{Str},\text{Cu}\}}\big) \supset \left\{\tilde{s} \in S_{\{\text{Str},\text{Cu}\}} : \phi_{\text{Str}}(\tilde{s}) < \epsilon_{\{\text{Str},\text{Cu}\}\to\{\text{Cu}\}}\right\} \\
&\quad = [0, 120] \times \left(\frac{5 - 2\epsilon_{\{\text{Str},\text{Cu}\}\to\{\text{Cu}\}}}{8}L, \frac{3}{4}L\right)
\end{aligned}$$

From Definition 34 we have

$$\begin{aligned}
&\text{Dom}\big({}_{\{\text{Str},\text{Cu}\}}\text{inc}_{\{\text{Str}\}}\big) \supset \left\{\tilde{s} \in S_{\{\text{Str}\}} : \pi_{\{\text{Str}\}} \le \phi_{\{\text{Str}\}}(\tilde{s}) \text{ and } \phi_{\{\text{Cu}\}}(\tilde{s}) > 0\right\} \\
&\quad = [0, 120] \times \left(\frac{3}{8}L, \frac{5 - 2\pi_{\{\text{Str}\}}}{8}L\right] \\
&\text{Dom}\big({}_{\{\text{Str},\text{Cu}\}}\text{inc}_{\{\text{Cu}\}}\big) \supset \left\{\tilde{s} \in S_{\{\text{Cu}\}} : \pi_{\{\text{Cu}\}} \le \phi_{\{\text{Cu}\}}(\tilde{s}) \text{ and } \phi_{\{\text{Str}\}}(\tilde{s}) > 0\right\} \\
&\quad = [0, 120] \times \left[\frac{3 + 2\pi_{\{\text{Cu}\}}}{8}L, \frac{5}{8}L\right)
\end{aligned}$$

7.4. The algorithms for the three modes

Recall the rough templates proposed in subsection 4.2. The algorithm $\mathcal{B}_{\{\text{Str}\}}$ is

```
declaration state : S_{Str}, v : speed_{Str}, x : position_{Str}, checkTransfer, checkSpeed : check
state := (0, 0)    % set presumed beginning state
x := O_pos(0)    % x is now the current position from the environment
state := updatePosition(x, state)    % update the state with the measured x
v := 120
checkSpeed := O_power(v)    % we choose not to check for exceptions for the power controller
state := updateSpeed(v, state)    % update the state with our velocity setting (not measured)
loop
  x := O_pos(0)    % x is now the current position from the environment
  state := updatePosition(x, state)    % update state with new position
  if φ_X(state, Str) < 9/10 then checkTransfer := tran(state, {Str, Cu}) else checkTransfer := OK
  if checkTransfer = NotOK then checkSpeed := O_power(0)    % if mode transfer fails, stop car
return
```

[4] We could have made things pointlessly more difficult by having distances on the straight and on the curves measured by different units (linear v polar; imperial v metric units). Programmers working on real world systems will be familiar with such legacy issues.

The algorithm $\mathcal{B}_{\{\text{Str,Cu}\}}$ is

```
declaration state : S_{Str,Cu}, v : speed_{Str,Cu}, x : position_{Str,Cu}, checkTransfer, checkSpeed : check, t : [0, 1]
input state    % the state is transferred from {Str}.
loop
  x := O_pos(0)    % x is now the current position from the environment
  state := updatePosition(x, state)    % update state with new position
  t := φ_X(state, Cu)
  v := 80 × t + 120 × (1 − t)
  checkSpeed := O_power(v)    % Implement a gradual slow down to 80 km/hr
  if t = 1 then checkTransfer := tran(state, {Cu}) else checkTransfer := OK
  if checkTransfer = NotOK then checkSpeed := O_power(0)    % if mode transfer fails, stop car
return
```

The algorithm $\mathcal{B}_{\{\text{Cu}\}}$ is

```
declaration state : S_{Cu}, v : speed_{Cu}, checkSpeed : check
input state    % the state is transferred from {Str, Cu}.
v := 80
checkSpeed := O_power(v)    % we choose not to check for exceptions for the power controller
```

7.5. A product system and introducing interactions

The simplicial complex $\mathcal{C}$ for one car for our track in Fig. 10 is quite simple, it is the 1-simplex on the left in Fig. 12. To make this more interesting we could consider two cars (car 1 and car 2) on the track. The reader may remember toy racing tracks where each car had a separate slot, so that the cars could not collide. In this case there is no need for the cars to interact, they can be controlled entirely separately, and then we get the product $\mathcal{C}_1 \times \mathcal{C}_2$ of the simplicial complexes for one car, which is the 3-simplex (solid tetrahedron) on the right in Fig. 12. There the vertex (Str, Cu) corresponds to car 1 being in mode {Str} and car 2 being in mode {Cu}. Then $\mathcal{C}_1 \times \mathcal{C}_2$ consists of all subsets of

$$\big\{(\text{S}, \text{S}), (\text{S}, \text{Cu}), (\text{Cu}, \text{S}), (\text{Cu}, \text{Cu})\big\}$$

and the partition of unity $\psi : [0, L] \times [0, L] \to \Delta_{\mathcal{C}_1 \times \mathcal{C}_2}$ is given by

$$\begin{aligned}\psi(x_1, x_2) = {} & \phi_{\text{Cu}}(x_1)\,\phi_{\text{Cu}}(x_2)\,e_{(\text{Cu},\text{Cu})} + (1 - \phi_{\text{Cu}}(x_1))\,\phi_{\text{Cu}}(x_2)\,e_{(\text{S},\text{Cu})} \\ & + \phi_{\text{Cu}}(x_1)\,(1 - \phi_{\text{Cu}}(x_2))\,e_{(\text{Cu},\text{S})} + (1 - \phi_{\text{Cu}}(x_1))\,(1 - \phi_{\text{Cu}}(x_2))\,e_{(\text{S},\text{S})}\end{aligned}$$

For simplicity, here and later, we suppress mention of the velocity in the state space and only consider the position coordinates $(x_1, x_2) \in [0, L] \times [0, L]$, where x_i is the position of car i.

To introduce interaction between the cars we use a chicane where the track narrows so that the cars can collide, as shown in Fig. 13. Now in the mode (Str, Str) in the product and only in that mode we have to allow for interaction. In any other position in the 3-simplex we have at least one car either in the curve or in the region where the straight and curve are both in play near position $L/2$, and thus at least one car is beyond the chicane so we have no concerns about a collision.

In the $[0, L] \times [0, L]$ picture of the state space the chicane appears as a forbidden square (shaded in the left picture in Fig. 14), where we do not allow both cars to be in the chicane at the same time. As the cars cannot reverse, their paths move up and to the right as time increases. The simplest way to avoid the square is to place a shield downwards and to the left to deflect incoming paths away from the square, and that is what we show in the left picture in Fig. 14, where we have magnified the region around the chicane. This arrowhead shield is made from the union of two overlapping rectangles, which are marked with W in the two rightmost pictures in Fig. 14.

The rightmost pictures in Fig. 14 are two covers of the joint state space $[0, L] \times [0, L]$ (we have suppressed the velocity). The centre picture is the cover for car 1, and the rightmost for car 2. We beg the reader's indulgence for not showing

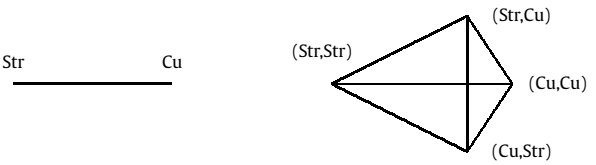


Fig. 12. The simplicial complex $\mathcal{C}$ for one car (left) and $\mathcal{C}_1 \times \mathcal{C}_2$ for two cars (right).

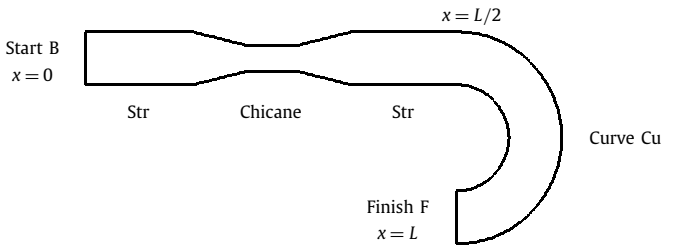


Fig. 13. A track for two racing cars with a chicane.

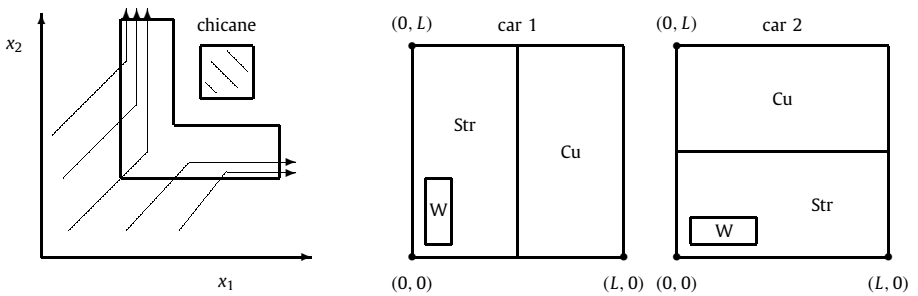


Fig. 14. Avoiding a collision at the chicane.

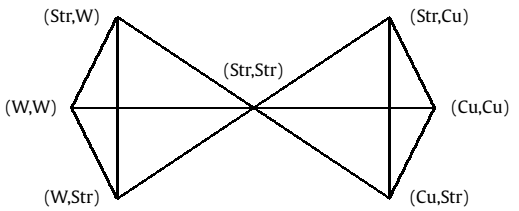


Fig. 15. The simplicial complex for two cars with a chicane, two solid tetrahedra meeting at a point.

the overlaps between the subsets in the cover, but the pictures were complicated enough already. The dynamics needed to shield the forbidden chicane square are now quite simple. If the path of the joint system in the cover for car i enters the subset W (for Wait), then car i stops, and does not start until the path leaves W. This is illustrated in the outside four paths entering the arrowhead in the leftmost picture. The only complication is what happens when the path enters the intersection of the car 1 and car 2 W subsets, i.e., the (W,W) mode of the joint system. Then, according to the previous instruction, both cars will come to a halt. The simplest thing is to have a timer on the cars to restart them, with the timer on car 2 being shorter so that it starts first. This is shown on the middle path entering the arrowhead, where both cars halt for a time when the path changes direction.

The complete simplicial complex for our two car system is shown in Fig. 15. On the right we have the product simplex from Fig. 12. This has now been altered by changing only the (Str, Str) mode, and now we join it to the 3-simplex which is the product complex for two copies of the {Str, W} complex. However this second (leftmost) 3-simplex does not simply have the product algorithm (i.e., two algorithms for each car running completely independently). Here the algorithm for car i has to track the position of the other car, and we also have the different timing for the car i W algorithms.

If the reader looks again at Fig. 14 they will see that we have been rather careless, we have no subset to take care of the forbidden square where two cars are simultaneously in the chicane. If that were to happen, the control systems would simply assume that they were both in mode Str, and disaster would likely ensue. However, that cannot happen, as our arrowhead shield means that the cars could never be in that position, is that not so? We note: "No plan survives contact with the enemy."[5]

[5] After Helmuth von Moltke the Elder (1800-1891).

8. Connections with hybrid systems and specification and correctness

The physical systems we have used to motivate and explain the mathematical framework deserve some elaboration. There are several established (essentially self-contained) theories for modelling such systems, such as smooth dynamical systems, linear and nonlinear control systems, and hybrid systems. Furthermore, the role of our framework for algebraic and logical methods for system design deserve comment. Some of these topics we can address now, though some fall into the category of directions for further work.

8.1. Connections with theories about hybrid systems

Over the years, the term hybrid system has had a number of meanings, such as simply combining analogue and digital computation [22]. To engineers, the term usually refers to a type of control system with analogue and digital behaviour [16]. We prefer to speak of an analogue-digital system, being a system in which a continuous physical component, process or environment is monitored and governed by a discrete algorithmic process.[6] This simple description covers a large range of systems from control systems for machines, industrial plant and buildings to systems that monitor and surveil people. As the ways of studying such systems are many and growing it is wise not to be too precious about the term hybrid system (cf. [14]).

However, in the formal methods community, hybrid systems do seem to have an expected or default interpretation. Hybrid systems have an established theory, software tools for specification and verification, and regular application areas – all of which are destined to grow as applications expand and diversify, and the practicality of methods improve and become better known.

The computational aspects of designing hybrid systems received a great impetus in the 1990s from studies by Zohar Manna, Amir Pnueli and Tom Henzinger [17,12], who developed formal methods for modelling, based on finite automata and differential equations, and for reasoning, based on the use of modal and temporal logics and model checking. Thus, for some readers the term hybrid systems suggests work on systems based on finite state automata of various kinds. We will summarise some features of the formal methods for modelling to help clarify the ways our approach is distinct.

Hybrid systems are studied by Petri nets or hybrid automata. In their representation as planar graphs, the nodes in the hybrid automata are called *locations* or *finite states*. Terminology varies in the literature – the finite states in the automaton have been known to be called modes. The nodes can have *invariants* – conditions which must be satisfied as long as the system is in that state, and if they stop being satisfied, then the system must move to another state. The arrows between the nodes represent the allowed state transitions and have *guard conditions*, which must be satisfied for the system to move along that arrow. The arrows can also have associated *actions*, which specify instantaneous (or nearly so) changes of variables on taking the arrow. The states typically have equations for continuous changes of variables, e.g., differential equations.

In some software tools there can be hierarchies of finite states, where opening-up the node for one state reveals another hybrid automaton inside: the system Ptolemy [15] takes such a hierarchical approach with high and low levels of description; the top level components are referred to as modes. A mature introduction to hybrid systems and logics for verifying hybrid systems is [18]; it explains a verification tool – KeYmaera – and two interesting case studies. Platzer's viewpoint on the importance of modelling the physical aspects of a hybrid system is well expressed. Platzer uses the term mode for state of an automaton.

In contrast, our key concept of a family of modes is intended to start the process of system design by categorising the *physical* behaviour of a system – separating *physical* concerns to use Dijkstra's celebrated phase. Technically, our modes can be designed separately with their own monitoring data and algorithms (which could be based on anything from a finite state automaton to a system of partial differential equations, or machine learning predictive models from regressions and decision trees to neural nets).

Viewpoint. In our thinking about analogue-digital systems, it is the real world component that is 'in control' – the software can only *respond* to data about the physical behaviour. So classifying the physical behaviours is important. Each mode is independent and sophisticated and comes with an externally imposed set of objectives and orders. Theoretically, the modes are determined by sets of states of the system; thus, physical behaviour as classified by modes determines the structure of a software specification at the highest level. For each mode α we have a set of data S_α necessary to describe the system. We also have rules for its evolution and (subtle) notions of modes of behaviour overlap, which enables transitions between modes. The mode evaluation function evaluates how well the current mode is performing as its behaviour changes in time – whether it is 'fit for purpose'.

Nondeterminism. A hybrid automaton *may* be nondeterministic, meaning that more than one arrow from a node may have its guard condition satisfied. For our modes, however, the overlap structure and the idea that all information has errors *forces* a high degree of nondeterminism that must be a focus of attention and be resolved. In fact, where a mode transition has two choices we have three modes having a simultaneous intersection, and can expect a whole continuum of possible paths

[6] We have long studied the nature and computational power of such analogue-digital systems [1,6,23,24,5,7].

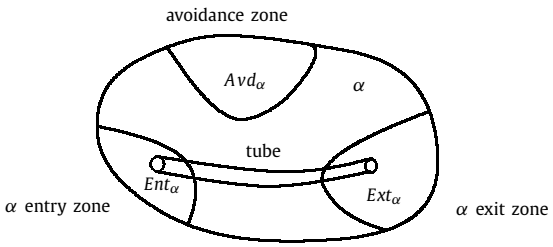


Fig. 16. A safety and liveness problem in mode α.

ranging between two new modes, each being a plausible path to take. This resolution is a focus of the semantic modelling, especially the alerts and interventions defined by κ, π and the mode evaluation function.

Visualisation. To tackle the transitions we construct a geometric representation of the system's whole state space, expressing the overlapping of the subsets defining the modes. Thus, choice of modes is explicitly represented by the geometry of the simplicial complex in $\mathbb{R}^n$. From a topological point of view, it corresponds to homotopic paths through state space. The continuum of possibilities in evaluating the arrows is encapsulated into a mode evaluation function, which lies at the heart of our approach.

As we have noted earlier, visualisation in a simplicial complex constitutes a high-dimensional generalisation of the planar graph visualisation of other discrete models such as automata.

8.2. Connections with specification and correctness

Our approach makes a distinction between the physical and the computational worlds. To create the software for an analogue-digital system, the physical components must be represented by an abstract specification that constitutes a data type interface between physical quantities and software. The specification and validation of an analogue-digital system presents certain problems outside software engineering as the reliability of an analogue-digital system depends upon the abstract assumptions made about the physical system. The design space between the real world system and a formal specification is a focus of our theorising.

Ideally, this introduces into theoretical consideration five component ideas about data:

- actual physical behaviour of a system;
- measurements of actual physical behaviour;
- predictions about behaviour by mathematical models;
- computations about behaviour;
- unexpected behaviour.

The concepts, principles and mathematical models are upstream from specific formal methods and their languages and tools. Our models can have many algebraic and logical encodings. We hope that existing and possibly new constructs would emerge to capture the various notions we encounter in say domain specific software development: physical behaviour as a semantic topic; local independence of modes; overlapping and superposition; re-construction of global from local behaviours; role of measurement and monitoring data, alerts and interventions; and fuzziness in decisions to transition.

Moving downstream – the transition to formal methods for programming – we need to specify formally criteria to access the modes, when to change mode, and how to choose a new mode. There is no shortage of formal techniques to try. To illustrate, we can start with looking at how the system enters and leaves a mode, keeping in mind possible liveness and safety properties.

For each mode α we must specify an entry zone Ent_α and exit zone Ext_α local to α; we might also specify a set Avd_α of states to be avoided. See Fig. 16.

These three sets can be approximated by specifications in a logical language. We can turn to Floyd-Hoare triples: in the classic notation, $\{p\}S\{q\}$ asserts that on executing program S on all input states satisfying precondition p, on termination of S, the output final states satisfies postcondition q. This idea has several generalisations and roles depending on the nature of the conditions p and q and the termination of S. One purpose of triples is to give someone who is not familiar with the workings of the program S enough information to correctly integrate it with another program in a manner allowing, or requiring, ignorance of the implementation.

For the purpose of mode transition, composition of pre and postconditions can be used: a postcondition for one mode can imply the precondition for another mode. This is just the Rule of Consequence:

$$\{p\}S_1\{q'\} \quad \text{and} \quad \{p'\}S_2\{q\} \quad \text{and} \quad q' \Longrightarrow p' \text{ implies } \{p\}S_1; S_2\{q\}.$$

What information is needed to specify the pre and post conditions? The pre and post conditions define subsets of Ent_α, Ext_α and Avd_α. Since we expect there to be a number of possible choices for modes, there can be a number of such specifications, at least one for each possible new mode.

Since we are thinking about actual physical behaviour, for algorithms and programs the answer is not so simple. Although the states and behaviour of the program are defined by the semantics of the programming language, running code is further determined by properties of the language implementation and the computer system. The essential point is that after measurement is delivered via oracles, all the information that is relevant, or can be known, lies within the software.

The choice of new mode on leaving a mode requires more formulae to arbitrate between the relevant options. Thus, the pre and postconditions will acquire an internal structure that expresses several more tasks. These connections are pointers to further work.

9. Developing the model

To develop the model, we will need a variety of case studies. However, some existing mathematical theories offer ready-made general models to help develop and test new theory about our model. For example, our approach is close to the theory of dynamical systems based upon smooth manifolds, which is a potential general source of examples and advanced modelling tools – see below.

9.1. Summary

Our task was to create a mathematical model of a complex analogue-digital system that operates in a finite number of distinct physical modes. Our novel idea is to model the structure of a family of modes by an abstract simplicial complex $\mathcal{C}$. To bring together the modes to create a model of the whole system, we have used the abstract simplicial complex $\mathcal{C}$ to derive appropriate sheaves containing the key components of the modes $X \in \mathcal{C}$, namely: the packages D_X with their state spaces S_X, data types $\mathcal{D}_X$ and algorithms $\mathcal{B}_X$ and the maps needed for mode transitions. To formally quantify – and visualise geometrically – we have used the abstract simplicial complex to make a concrete simplicial complex $\Delta_\mathcal{C} \subset \mathbb{R}^n$ containing a simplex Δ_X for each mode $X \in \mathcal{C}$. Functions ϕ_X that evaluate states, and alert thresholds κ, π that trigger transitions, complete the quantification.

As the framework stands, there are a number of developments that will bring it closer to established theoretical work on formal methods for specification and program correctness. We have highlighted hybrid systems and correctness logics in the previous section. In our exposition, we have held back on using the algebraic theory of data types, which contains many notions relevant to local and global specification and computation. The data types that make up the framework are computable structures – typically, based on measurements by rational numbers. The interface with a real world can be analysed by the theory of *physical oracles* [5]; in dealing with the real world we must allow for issues like precision, errors and delays.

The heart of our analysis of the digital control of a physical system is the dichotomy between the digital world – which is the only one that the software knows anything about – and the real world – which is the only one that actually matters.

9.2. Mathematical connections: modes and manifolds

In applied mathematics, the state space of many physical systems is often idealised as an n-dimensional smooth manifold, which is a topological space which has a family of local coordinates $(x_1, \ldots x_n) \in \mathbb{R}^n$ and allows differentiability ([2,3]). These local coordinate systems are maps that take a covering of open sets of the space into $\mathbb{R}^n$ and are called *charts*. In certain situations, each chart can be designed to be the basis of a mode in our sense.

Consider a system whose global state space is a manifold M. A behaviour of the system can be thought of as a point moving on path in the manifold M. A mode transition is the passage of the point from one chart (= local coordinate system) to another.

Let $S_\alpha \subset \mathbb{R}^n$ be the set of coordinates for chart U_α and $S_\beta \subset \mathbb{R}^n$ to be the set of coordinates for chart U_β; let the coordinates be denoted x^i and y^i for $1 \leq i \leq n$, respectively. Then we have sets of coordinates corresponding to the intersection $U_\alpha \cap U_\beta$ that are the subsets $I_{\alpha\beta} \subset S_\alpha \subset \mathbb{R}^n$ and $I_{\beta\alpha} \subset S_\beta \subset \mathbb{R}^n$; and there is a smooth transition function $\tau_{\beta\alpha} : I_{\alpha\beta} \to I_{\beta\alpha}$ with inverse $\tau_{\alpha\beta} : I_{\beta\alpha} \to I_{\alpha\beta}$.

In terms of our model, now we set $S_{\{\alpha,\beta\}} = I_{\alpha\beta} \times I_{\beta\alpha}$. On moving from U_α into the intersection $U_\alpha \cap U_\beta$ (assuming it is non-empty) we take $(x_1 \ldots x_n) \in I_{\alpha\beta} \subset S_\alpha$ and copy it into $S_{\{\alpha,\beta\}}$, and then use the change of coordinate map to calculate the y^is, giving $\big((x_1 \ldots x_n), (y_1 \ldots y_n)\big) \in S_{\{\alpha,\beta\}}$. Note that we only have a partial map from S_α into $S_{\{\alpha,\beta\}}$.

9.3. Widening the scope of applications

Another direction for further research is to widen the scope of thinking in terms of modes. The model is a formalisation of decision making using abstract and concrete simplicial complexes to classify and quantify evidence. Decision making permeates computing applications. Many decisions in applications involve weighing up data, arguments, evidence etc. and making a judgement. Probability theories have helped analyse decision making since the 18th Century, especially in the

birth of actuarial mathematics. More recently, belief theories [20], many valued logics [9], neural nets and sundry other machine learning techniques, have provided mathematical theories with which to explore inexact reasoning and making decisions. The scope of the general idea of modes could become much wider, guided by formalisation using an abstract and concrete simplicial complexes.

Human systems: The intuitive ideas about modes, first illustrated and expounded in Section 2, can be extended from examples of 'hard' physical systems to 'soft' human systems. To give a flavour of modes and mode transitions involving people, consider this example:

Example 38. Consider a patient with disease α being monitored in a hospital. We can suppose that $S_{\{\alpha\}}$ is the information held by the hospital about the patient. At some stage the condition of the patient changes to indicate that the patient may be developing disease β. Monitoring the patient is now in a mode $\{\alpha, \beta\}$ where tests related to both diseases have to be done, and where complications may arise which would not occur in a patient with a single infection of either disease. The simplest way for the hospital to accommodate the newly required information is simply to make $S_{\{\alpha,\beta\}}$ strictly contain $S_{\{\alpha\}}$.

Now we take a patient in mode $\{\alpha, \beta\}$ (i.e., presumed to have both conditions α and β). To move to mode β we require evidence that the patient no longer has condition α. In that case we would have a map $S_{\{\alpha,\beta\}} \to S_\beta$ which would retain the medical history, but delete the flags for current monitoring and medication specific to condition α.

Typologies for modes: We can use a simplicial map from the simplicial complex $\mathcal{C}$ of modes to another simplicial complex $\mathcal{E}$ to classify certain aspects of the system. One example is for $\mathcal{E}$ to represent security clearance, and the simplicial map then maps access modes to security clearances. Recall that partially ordered sets have appeared in models of security and information flow [10]. Different simplicial maps could be used to classify other behaviours of the system or restrict access to oracles, for example in a social system distinguishing between managerial hierarchies and the authority to rewrite regulations. In this manner modes could inherit behaviour from multiple classifying maps in a transparent fashion specified at the system design stage.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] T. Ambaram, E.J. Beggs, J.F. Costa, D. Poças, J.V. Tucker, An analogue-digital model of computation: Turing machines with physical oracles, in: A. Adamatzky (Ed.), Advances in Unconventional Computing vol. 1: Theory, Emergence, in: Complexity and Computation Series, vol. 22, Springer-Verlag, 2016, pp. 73–115.
[2] V.I. Arnold, Ordinary Differential Equations, MIT Press, 1973.
[3] V.I. Arnold, Mathematical Methods of Classical Mechanics, Springer-Verlag, 1978.
[4] R. Baduel, J-M. Bruel, I. Ober, E. Doba, Definition of states and modes as general concepts for system design and validation, in: 12e Conference Internationale de Modelisation, Optimisation et Simulation, MOSIM 2018, 27 June 2018 - 29 June 2018, Toulouse, France, 2018.
[5] E.J. Beggs, J.F. Costa, J.V. Tucker, Axiomatising physical experiments as oracles to algorithms, Philos. Trans. R. Soc. A 370 (2012) 3359–3384.
[6] E.J. Beggs, J.F. Costa, D. Poças, J.V. Tucker, An analog-digital Church-Turing thesis, Int. J. Found. Comput. Sci. 25 (2014) 373–389.
[7] E.J. Beggs, J.F. Costa, J.V. Tucker, Three forms of physical measurement and their computability, Rev. Symb. Log. 7 (4) (2014) 618–646.
[8] E.J. Beggs, J.V. Tucker, Analogue-digital systems with modes of physical behaviour, arXiv:1412.2643, 2014.
[9] L. Bolc, P. Borowik, Many-Valued Logics. 1. Theoretical Foundations, Springer-Verlag, 1992.
[10] D. Denning, A lattice model for secure information flow, Commun. ACM 19 (5) (1976) 236–243.
[11] EXOMARS 2016 - Schiaparelli anomaly inquiry, ESA 2017. Link: http://exploration.esa.int/mars/59176-exomars-2016-schiaparelli-anomaly-inquiry/#. (Accessed 18 February 2022).
[12] T.A. Henzinger, The theory of hybrid automata, in: M. Kemal Inan, Robert P. Kurshan (Eds.), Verification of Digital and Hybrid Systems, in: NATO ASI Series, vol. 170, Springer, 2000, pp. 265–292.
[13] M. Heymann, F. Lin, G. Meyer, S. Resmerita, Analysis of Zeno behaviors in a class of hybrid systems, IEEE Trans. Autom. Control 50 (3) (2005) 376–383.
[14] G. Labinaz, M. Guay, Viability of Hybrid Systems, Springer, 2012.
[15] E.A. Lee, S. Tripakis, Modal models in Ptolemy, in: 3rd International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools, EOOLT 2010, Oslo, Norway, Linköping University Electronic Press, 2010.
[16] J. Lunze, F. Lamnabhi-Lagarrigue, Handbook of Hybrid Systems: Control Theory, Tools, Applications, Cambridge University Press, 2009.
[17] O. Maler, Amir Pnueli and the dawn of hybrid systems, in: Proceedings of the 13th ACM International Conference on Hybrid Systems: Computation and Control, ACM, 2010, pp. 293–295.
[18] A. Platzer, Logical Analysis of Hybrid Systems, Springer, 2010.
[19] Report on the Loss of the Mars Polar Lander and Deep Space 2 Missions, NASA JPL, Link: https://solarsystem.nasa.gov/missions/mars-polar-lander-deep-space-2/in-depth/, 2000. (Accessed 18 February 2022).
[20] G. Shafer, A Mathematical Theory of Evidence, Princeton University Press, 1976.
[21] E.H. Spanier, Algebraic Topology, McGraw-Hill, 1966.
[22] T.D. Truitt, Hybrid computation … What is it? …Who needs it?, in: IEEE Proceedings - Spring Joint Computer Conference, 1964.
[23] J.V. Tucker, J.I. Zucker, Computability of analog networks, Theor. Comput. Sci. 371 (2007) 115–146.
[24] J.V. Tucker, J.I. Zucker, Computability of operators on continuous and discrete time streams, Computability 3 (2014) 9–44.
[25] Y. Zhao, S. Maletić, Simplicial Complexes in Complex Systems: In Search for Alternatives, World Scientific, Singapore, 2021.