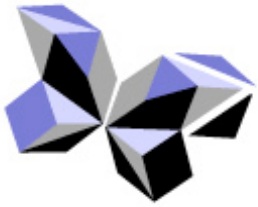


Digital Business Ecosystem

Contract n° 507953

Workpackage 18

Socio-economic perspectives of sustainability and dynamic specification of behaviour in Digital Business Ecosystems

Deliverable 18.4

Report on self-organisation from a dynamical systems and computer science viewpoint

Information Society Technologies

Project funded by the European Community under the “Information Society Technology” Programme

Contract Number: 507953

Project Acronym: DBE

Title: Digital Business Ecosystem

Deliverable N°: 18.4

Due date: 31/12/2006

Delivery Date : 05/03/2007

Short Description: This report discusses the need to support the emergence of two distinct reference theoretical frameworks underlying computing: a framework for software engineering that relies as much on language theory as on discrete mathematics and logic and that is well-suited to address (non-exclusively) functional requirements, and a framework for computer science that relies on mathematics and logics, that is enriched by biological metaphors and models, and that is well-suited to address (predominantly) non-functional requirements.

Partners owning: Paolo Dini (LSE)

Partners contributed:

Made available to: unlimited distribution

Versioning		
Version	**Date**	**Author, Organisation**
1	4 Jan 06	Paolo Dini, LSE
2	30 August 06	Paolo Dini, LSE
3	5 March 07	Paolo Dini, LSE
4		

Quality check

1st Internal Reviewer: Paul Krause (UniS)

2nd Internal Reviewer: Angelo Corallo (ISUFI)

Executive Summary

This report discusses the need to support the emergence of two distinct reference theoretical frameworks underlying computing: a framework for software engineering that relies as much on language theory as on discrete mathematics and logic and that is well-suited to address (non-exclusively) functional requirements, and a framework for computer science that relies on mathematics and logics, that is enriched by biological metaphors and models, and that aims to relieve the burden of designing for non-functional requirements. In the context of the latter, the report discusses how the Turing Machine model of computation cannot model self-organisation, and shows how more recent theories as well as software and network engineering practice are leading to the emergence of a new model, generally referred to as the Interaction Machine. We discuss how the Interaction Machine could be seen as resulting from the on-going convergence of the two research fields of bio-computing and computational biology. Thirdly, we focus on what we consider an embryonic instance of Interaction Machine, the Fraglets, and show how they could potentially embody elements of automata theory, process algebras, multiset rewriting methods, and dynamical systems theory.

Table of Contents

1. INTRODUCTION

This report continues the investigation begun in D18.1 toward an understanding of self-organising processes that can be useful to software systems and to software engineering in particular. The research discussed in this report is mainly focused on the technical and scientific aspects of digital ecosystems and describes work performed in Tasks S22-"Theoretical and mathematical investigation of self-organisation in natural and social science" and S23-"Self-organisation, DNA, and formal systems", within WP18, during Year 2 and Year 3 of the project.

New workpackage name

WP18 was originally named "DBE UML Profile". During the course of the project it became increasingly clear that the problems being addressed were much too deep and broad to be able to develop within the 3 years of the project and in collaboration with Soluta a software engineering modelling language based on the research findings. More importantly, the theoretical framework that is starting to take shape to connect software specifications to running programs appears to be fundamentally incompatible with UML. This is because UML is chiefly concerned with the *functional requirements* of a software application, and with deriving the application's structure from such requirements. Biologically-inspired self-organising processes, on the other hand, are likely to be of greater relevance to the *non-functional requirements* of the same application. Although the latter can in principle be employed in support of the former, transparently to the user, the theoretical starting point remains mathematics and biology rather than UML. UML is the end-point that might be reached after a host of difficult computer science problems have been solved. This research aims to uncover these problems and therefore is biased distinctly more toward mathematics and computer science than toward software engineering.

Between the writing of the project proposal and the start of the project the opportunity to involve LSE social scientists in this workpackage materialised, whose work is reported in Chapter 3 of D18.1 and in the whole of D18.3 and D18.7. As a consequence it seems appropriate to rename the workpackage in a way the reflects more accurately the research we are actually performing. The new name and the broader emphasis can also be understood to lay the groundwork for much of the objective (i.e. non-reflexive) research the OPAALS project[1] will perform.

One half of WP18 addresses the formalisation of biologically-inspired mechanisms into computational models first, and ultimately into architectures, algorithms, and data structures that can lead to the *adaptation and self-organisation* of software in a dynamic run-time environment that is ultimately driven by user requests and user behaviour. This part of the work is conceptualised within the naturalist/functionalist philosophical tradition. The other half of the workpackage analyses the socio-economic interactions present in open source communities with the aim to propose policies that individual, institutional, and commercial actors can operationalise to support the formation of *sustainable* digital business ecosystems. This part of the work is conceptualised within the hermeneutic, social constructivist, and other social science traditions. Although these two traditions are at present generally considered to be incommensurable,[2] and although these two sets of WP18 research activities address completely different biological/computing and socio-economic processes, they are both concerned with the contributions these very different areas of research can make, in correspondingly different ways, to the sustainability of ICT systems and of digital business ecosystems in particular. The new title of WP18 has therefore been chosen to highlight what we regard as the most important sources of sustainability of DBEs in the very different senses this term assumes in the two different contexts: "Socio-economic perspectives of sustainability and dynamic specification of behaviour in Digital Business Ecosystems".

[1] The OPAALS Network of Excellence started on 1 June 2006 and will last for 4 years: www.opaals.org
[2] See D18.1 and D4.1.

Sustainability of the business ecosystem is thus seen to depend on the integration of social interactions and dynamics (including language and the formalisation of knowledge) with economic and business models. Sustainability from the point of view of self-organisation, learning and adaptation of the software infrastructure and services, on the other hand, can be regarded as a non-functional requirement of DBE software.

Definitions and scope

The DBE system is shown in Table 1.1, which distinguishes between the "local" and the "distributed" view of the DBE and its sub-systems. The top row is concerned with the business enviroment, it is methodologically closest to social science, and is not directly addressed by this report, although Chapter 2 does address social science questions at a theoretical level. The second row is concerned with the service creation environment and it closest to software engineering. One of the outputs of WP18 is the realisation that software engineering has as much in common with language as it does with the discrete mathematics underlying computer science. The impact on the second row of the research described in this report, therefore, is at this point still relatively small. The third row is concerned with the execution environment of the DBE, which relies on a combination of networking and software technologies. The greater interactive character of the ExE brings it closer to concurrency theory and to the topics discussed in this report. Finally, the fourth row is concerned with the evolutionary environment of the DBE. The topics discussed in this report are closest to the EvE among all the DBE components; however, the report aims to develop a theory of gene expression-inspired computing that can be seen as complementary to evolutionary computing.

Table 1.1 System and component view of DBE

<table>
<tr><th>Local view</th><th colspan="3">Distributed view</th></tr>
<tr><td>SME</td><td colspan="2">Business Ecosystem</td><td rowspan="4">Digital Business Ecosystem</td></tr>
<tr><td>DBE Studio</td><td colspan="2">Service Factory (SF)</td></tr>
<tr><td>ServENT[3]</td><td>Execution Environment (ExE)</td><td rowspan="2">Digital Ecosystem (Run-time "Artificial Life" system)</td></tr>
<tr><td>Habitat</td><td>Evolutionary Environment (EvE)</td><td>Digital Business Ecosystem (All the infrastructure, models, & services)</td></tr>
</table>

Another row above the business ecosystem could in principle be added, related to the socio-economic environment, the regional economies, and the Knowledge Economy, but that would over-extend the validity of the "component view" of the DBE, which is better suited to describing technology than social, political, and economic institutions.

The above definitions work well within the DBE Project, but the term "digital ecosystem" is overloaded. It can both mean an artificial life system deeply buried inside the DBE, or it can also refer to a broader meta-concept that can be instantiated in different ways depending on different socio-economic or cultural contexts. For example:

- digital ecosystem for B2B business transactions and collaboration between SMEs: DBE
- digital ecosystem as a knowledge creation and sharing platform: OPAALS Open Knowledge Space
- digital ecosystem for distributed P2P gaming (does not exist yet)
- digital ecosystem for agriculture: Indian DEAL project http://emandi.mla.iitk.ac.in/deal/
- digital ecosystem as a marketplace
- etc…

All these ecosystems are assumed to be interconnected and interdependent to varying degrees.

[3] ServENT = SERV[er] + [cli]ENT

Objectives

Top-down model-driven development or bottom-up emergence of behaviour?
The principal long-term objective of WP18 was to contribute to the development of a software engineering framework that will facilitate the specification, the design, and the construction of digital ecosystems. Implicit in this statement is the mutual dependence between the concept of desired behaviour of an artifact and the concept of its abstract representation, or model. Because digital ecosystems by assumption are self-organising, their development appears to give rise to a contradiction that undermines this objective: according to the strict definition of "self-organising", the desired behaviour of the DBE would appear to be *not to require an abstract model*. This extreme view was never thought to be feasible and at the end of the project is now understood to be incorrect or, more appropriately, incomplete. Having established that a Digital Ecosystem is a socio-technical system, the presence and behaviour of the users become an integral part of the system. A functionalist, interpretive, phenomenological, critical, social-constructivist, or other theory that can be harnessed to inform, ultimately, policies for sustainable socio-economic development cannot help becoming of importance as central as engineering practice and mathematical formalism.

Top-down specification of functional behaviour, therefore, should not be regarded as an outmoded form of engineering to be replaced by biologically-inspired bottom-up emergence of structure and behaviour. Top-down specification of functional requirements can be seen as a form of language process, and therefore a social process. As such, it necessarily rejects the imposition of "absolute" laws of any kind, due to the fundamentally subjective, or perhaps inter-subjective, nature of social processes. This is reflected by the freedom of abstraction that software engineers treasure, and by their correspondingly strong aversion to any mathematical framework arbitrarity imposed. Software engineering seen as a socio-technical process, therefore, becomes as fundamentally important to the sustainability of digital ecosystems as the "nuts and bolts" of the infrastructure and the evolutionary character of the environment.

In hindsight, it is clear that one of the main objectives of WP18 was all-along to reconcile these two opposite viewpoints on computing. This objective has been achieved by recognising the legitimacy of a language-based view of software engineering for capturing functional requirements, peacefully coexisting with a mathematics-based[4] view of computer science for addressing the non-functional requirements of adaptability, self-healing, and self-organisation of the software.

The original objective has therefore been transformed by this research into a requirement to develop a mathematical framework that can accommodate unbounded adaptability of the services and components of the DBE without limiting the ability of the users to specify the functional behaviour they require. Judging from the infinite variety of life-forms seen in biology this objective seems feasible in principle.

Should WP18 focus on the infrastructure or the services?
Another important distinction is between the original focus of WP18, i.e. the DBE infrastructure as summarised in Table 1, and the life-cycle of the services that "inhabit" all the environments of the DBE. This distinction has become blurred, for three reasons.

The technical view. Firstly, because the design and implementation of the infrastructure is being achieved through traditional software engineering methods that have so far proven adequate. Thus if we focus on the technology being used today to build the DBE *infrastructure* it is not clear what "DBE-specific" features of the design process could be captured by, for instance, a UML profile for DBE. This has led us naturally to shift the focus of WP18 toward the self-organising features of the *services* as an area of the DBE that presents more immediate opportunities for applying biological principles and "biological design patterns" (Briscoe, 2002). Furthermore, and perhaps more significantly, the DBE is a service-oriented

[4] We are lumping somewhat loosely in the term "mathematics" much of what in computer science is referred to as "formal methods", which include concurrency theory and the many forms of logic, as well as any mapping into computer science from the mathematical theories underpinning physics and biology.

architecture, meaning that also the infrastructural components are designed according to the DBE Service architecture. For example, the Habitats, whose union forms the EvE, are implemented as DBE Services running in the ServENT like any other service.

The biological view. Secondly, because the increasing modularity and reuse of the DBE services (and of the software development methodology as a whole) cannot help but further blur the distinction between "service" and "infrastructure". This refers in particular to those basic services and atomic components that will get reused in many applications. Fig. 1.1 shows a parallel with the blurred distinction between species and environment found in biology. The point is that for any one layer of the "biology stack" all the layers below can be regarded as part of the environment, making the latter concept relative and not absolute. The ecosystem is the union of all the layers.

The business view. As explained in D5.1, the view of service combination and aggregation shown in Fig. 1.1 is overly simplistic. The most innovative aspects of the DBE, so far, have been found in the new possibilitites for collaboration and synergy between SMEs that it enables and supports, reinforcing the importance of social processes as intergral parts of digital ecosystems. This corresponds to the combination and aggregation of business models which, however, are not easily represented with simple graphics and coloured boxes. As discussed in D4.1, also business models belong to the category of "digital species" that live in the DBE, and should therefore also benefit from the principles of self-organisation we are developing. If *existing software applications and services* are combined through *innovative business models* enabled by the DBE, then clearly the self-organisation, if any, is happening at a level quite different to the technical. In this case the technical infrastructure remains simply an enabling medium for business communications in the sense of Winograd and Flores (1986).

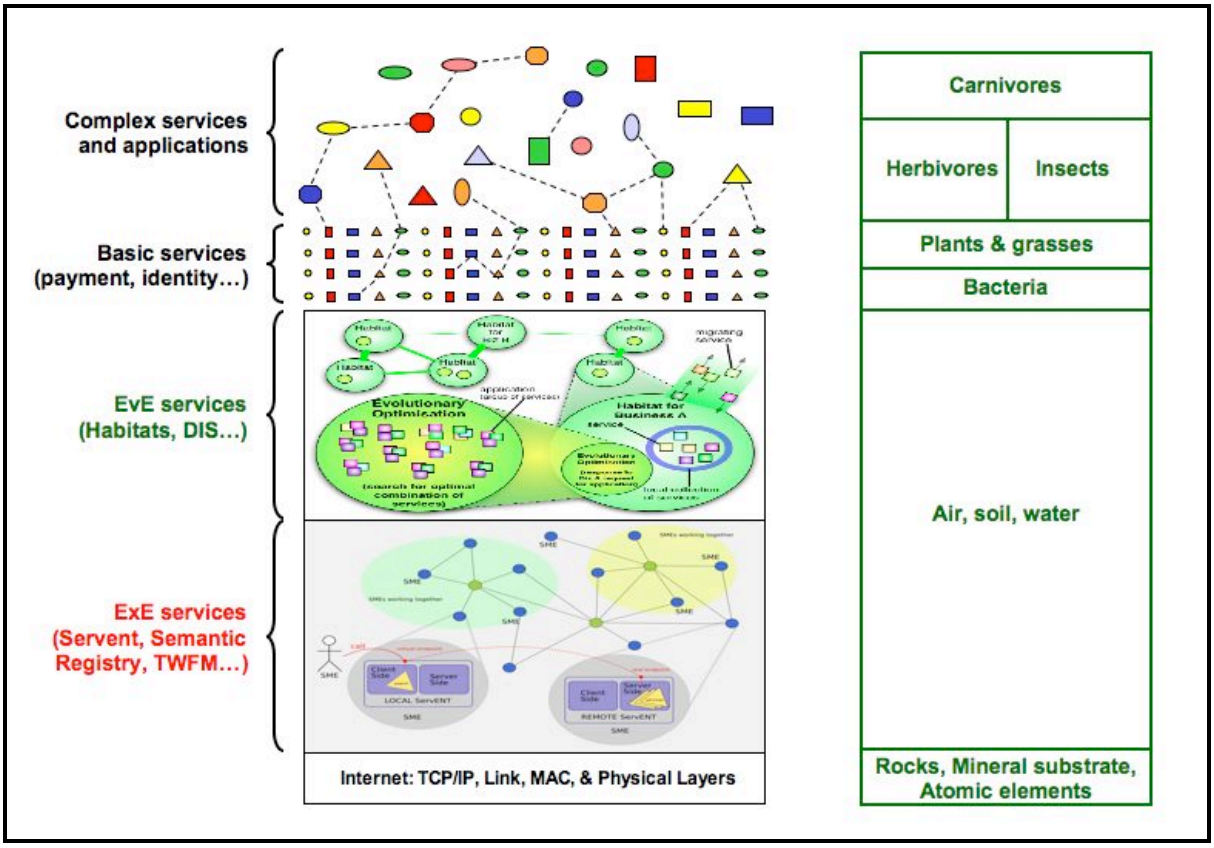


Fig. 1.1 Parallel of the fuzzy distinction between services/infrastructure & species/environment[5]

[5] The EvE diagram was created by Gerard Briscoe (HWU), the ExE diagram by Javier (Bob) Noguera (Sun).

Regardless of how the views on and capabilities of the DBE will evolve, WP18 addresses theoretical aspects of computing systems that ultimately transcend whether we are talking about services or infrastructure. We are therefore going to be opportunistic about which we focus on, with the understanding that in any case we are talking about instances of principles of wider generality.

The role of stability

Another objective of WP18 that has been stated consistently since the beginning of the project is to understand how to make dynamic software systems[6] of interacting components stable. This is believed to be a necessary criterion for such systems to be able to construct order. Even though the mathematical theory of non-linear dynamical systems seems useful for investigating the specific issue of their stability, there is a growing consensus (mainly in the cell biology research community) that its expressiveness is limited. It seemed preferable, therefore, to shift the attention away from non-linear dynamical systems until such time as the underlying mathematical theory is better understood. Even though the theory in question, that of Lie groups, is already considered well understood, it remains far from trivial to find suitable symmetries[7] that can solve a particular equation or set of equations; it is even more difficult to assign a physically meaningful interpretation to such regularities.

Regular structure and behaviour are, however, ubiquitous in biology (Thompson, 1917), such that a theory of self-organisation that cannot somehow account for Nature's symmetries would seem hopelessly incomplete. The work that was begun to analyse non-linear dynamical systems and Lie groups has therefore been retained as it is deemed important to help develop an intuitive understanding of this important and ubiquitous class of problems. Discrete groups and discrete symmetries were prioritised as the most likely theory that can formalise such regularities. It is well understood (as discussed more extensively in deliverable D18.1) that the regularities observed in physical and natural systems are a consequence of the regularities in the underlying physical laws. It is also well understood that no such laws exist in information systems, aside perhaps from Shannon's entropy theorem. The objective of this workpackage, therefore, was to formalise the *effects* of physical laws in the form of symmetries, without worrying about modelling the cause for such symmetries, and then impose the same mathematical framework on software systems, as an external and artificial constraint. This objective was not reached, but we report on the progress we have made to date in this direction. For example, we are starting to work with the theory of Galois Fields, which are built by adding algebraic structure to finite groups (Moschoyiannis, 2001). Galois Fields are used in network coding, for detecting and correcting errors in messages that have been purposedly extended with redundant bits. For such methods to work the manner in which redundancy is introduced cannot be arbitrary, and the correcting ability of the code in the presence of arbitrary errors is afforded by the intrinsic regularities of the Galois Field used to construct it.

As already explained in D18.1 by stability we are not referring to the elementary stability properties obtainable by linear stability analysis (Drazin and Reid, 1980) and embodied in the sign of the largest Lyapunov exponent (Bergé, Pomeau, Vidal, 1984). We are referring to dynamical systems that are "far from equilibrium" (see D4.1) and whose Lyapunov exponent analysis would yield positive exponents—and yet that remain within a particular subset of their accessible states. Biological systems fall in this category in their ability to remain "alive" within a finite range of allowed perturbations. In complexity science this property is sometimes described as the ability to retain a certain organisational structure that open complex-adaptive systems exhibit, called "organisational closure". Clearly, the generalised stability

[6] A clarification on word use and terminology is in order. In physics "*dynamical* system" refers to a system (of one or more degrees of freedom) the dependence of whose states on time is expressed mathematically through one or more differential or discrete "governing equations". Such governing equations are generally derivable from a Lagrangian and a minimisation principle such a Hamilton's principle. Shroedinger's equation of quantum mechanics and the low-dimensional non-linear dynamical systems that are discussed in this report fall in this category. In computer science and software engineering "*dynamic* system", on the other hand, refers to a software system or process that is able to react to changing conditions in its environment at run-time—as in the phrase "dynamic memory allocation". Because biology appears to lie at the intersection of these two views of the world, I will need to keep using both terms.

[7] As will be discussed in Chapter 4 symmetries are transformations that leave something invariant. In the case in question the invariance is with respect to the functional form of the ordinary differential equation(s) (ODE) being analysed.

that some low-dimensional non-linear dynamical systems exhibit can only be part of the answer since such systems are closed in every sense. Their appeal, which has justified studying such systems so far and may warrant additional study at a future time, comes from the fact that, for 2 or more degrees of freedom, each DoF can be treated as an open system interacting with the other DoFs.

An alternative view on "stable" self-organising systems is given by autopoiesis, although such an extension appears to be far from trivial. Autopoiesis literally means "self-generation" and therefore is clearly related to self-organisation. However, it is probably more accurate to say that it is a self-organisation "of higher order", meaning the self-organisation of a system that includes the self-organisation of its ability to recursively produce other self-organising copies of itself, like the infinite regression of facing mirrors, or of "recursive Russian dolls" (see Fig. 2.3).

This report, therefore, begins an exploration of the different ways in which open, complex, self-organising systems are able to retain a level of stability with their environment, even if not in equilibrium, that allows them to function reliably even if exhibiting a great deal of flexibility and robustness in the presence of unpredictable external perturbations. How such behaviour is achieved within the cell and how it could be mapped to software will remain a central concern of digital ecosystems research well beyond the DBE project.

Overview of Report

The report is a collection of research activities undertaken during Year 2 and Year 3 of the project. Some of these activities are connected and build on one another, others are disconnected and reflect different directions of enquiry that were attempted. Although they all aim toward the same goals discussed above, the level of integration is limited. More work will be necessary in OPAALS and other projects to carry forward the development of an integrated scientific/mathematical theory of digital ecosystems.

Chapter 2 has appeared as a chapter contribution to the DBE Book published by the European Commission (Dini, 2007). This chapter gives an overview of the scientific foundations of digital ecosystems research, highlighting important insights and outcomes. Chapter 3 summarises some of the activities that were begun to analyse low-dimensional non-linear dynamical systems. Chapter 4 provides a detailed discussion of the elementary concepts underlying continuous symmetries of differential equations, which will be applied to the non-linear problems of Chapter 3 at a later date. Chapter 5 draws some reflexive and methodological conclusions and outlines briefly the areas of research that are currently the focus of our attention.

2. A SCIENTIFIC FOUNDATION FOR DIGITAL ECOSYSTEMS

Preamble

One of the most theoretically challenging hypotheses of digital ecosystems research is that biological metaphors and isomorphic models can bring significant advantages to the e-Business software development process, to the self-optimising and self-healing properties of distributed run-time environments and service-oriented architectures, and to ICT adoption in general through greater adaptability of the software to user needs. While the concept of business ecosystem is not new (Moore, 1996) and the application of biological concepts and models to software has inspired whole fields of research, such as Artificial Life, their application to software in an e-Business context is quite innovative. Similarly, whereas the complementary roles companies play in value chains is reminiscent of the interdependence between species in biological ecosystems, using the ecosystem metaphor to conceptualise all the business, economic, and social relationships between all the private and public sector players and the academic/research institutions in a whole region is rather more ambitious. Less contentious, but still far from clear, is the role of Open Source (and more generally of a "Commons") in lowering the barrier of ICT adoption for small and medium-sized enterprises (SMEs), the role all these factors play in catalysing sustainable socio-economic development, and what regional policies can be derived from the research and engagement experience with SMEs. Further, the language-based and collaborative processes through which social networks of SMEs can influence their own sustainability in regional economies are not well understood, but ICTs are widely believed to offer the potential to amplify this social constructivist dynamic.

Overview

The paper will start with a brief discussion of the challenges posed by interdisciplinary research and will show how they largely originate from a dichotomy in how technology can be viewed from the points of view of social science and natural science. Following the social science perspective the paper will argue that the connections between language and technology lead to a recursively self-generating dynamic that is at the basis of the power of self-determination afforded by digital ecosystems. From this view informed by the connections between power and language this paper will *not* venture into an economics analysis beyond making cursory references to the interactions between the exchange economy and the gift economy (Berdou and Dini, 2005). The integration of these different accounts of social and economic action within a digital ecosystem theory is just now beginning to appear possible, but will undoubtedly require several more years of research to achieve an operational consensus supported by empirical evidence. The paper will instead use the inter-subjective view of the world common in some areas of social science, partly inspired by Heidegger's phenomenology, to follow Minger's characterisation of Maturana and Varela's work on autopoiesis as strongly relativist. This provides nothing as grand as a unification between social science and natural science (which is generally undesirable in any case) but, rather, an opportunity to recognise communications as a more modest "hinge" that connects these two rather incompatible disciplinary domains: regardless of how much the two individual domains will grow and evolve, in fact, it will always be possible to claim a strong overlap around concepts of inter-subjectivity and context-dependence of meaning, which characterise both social and biological systems.

Because a science of digital ecosystems promises at least a taxonomy of the fundamental concepts and principles, if not their integration, before surrendering to the allure of biology the line of argument will try to be reflexive and question the assumption that biology has anything at all to teach to computer science. In other words the paper will probe what the term "fundamental" could mean in the context of computer science and will fall back on biology only after a struggle that will hopefully appear convincing. This will finally bring us to the mathematically overwhelming but for some of us more familiar territory of the construction of order in biological and physical systems, of the current debates around possible explanatory theories thereof, and of the possible mappings of such mathematical theories to computer

science and software engineering, which might lend some credence to the ambitious claims of adaptability and evolvability made in the Preamble.

Social Science

Interdisciplinarity and reflexivity

The deeply interdisciplinary character of the digital ecosystems paradigm (that is, a body of theory combined with a community of practice and a set of research methodologies (Kuhn, 1996)) forces a recognition of the importance of the role played by the community or communities of research, if only because the barriers in communication across disciplinary domains cannot easily be ignored. Acknowledging the presence of the researchers amplifies the perception of the nature of knowledge as subjective, relativist and pluralistic. As a consequence, if the practitioners in the different fields are to establish a productive dialogue they cannot rely on unspoken assumptions, default scripts, and routine modes of interaction. They need to question themselves and their disciplines to find new contact points, which could be either theoretical or pragmatic. They cannot take themselves and the others for granted. In other words, they need to become reflexive.

As shown in Table 1.1, a simple example of the challenges involved is provided by the very definition of a digital ecosystem. Depending on the disciplinary domain, a rather different answer to the question "What is a digital ecosystem?" is assumed. The computer science definition is half-way between these two extremes. In fact, computer science has emerged relatively recently, as a lenticular image[8] posing an intriguing puzzle in its shifting identity between the world of technology and the world of people.

Table 2.1 A Digital Ecosystem is…

Social Science	***Computer Science***	***Natural Science***
• A community of users • A shared set of languages • A set of regulatory norms and guidelines to foster trust • A population of services • An open-source service-oriented infrastructure	• Several categories of users • A set of formal languages • A security and identity infrastructure • A service-oriented architecture • A service development environment • A distributed P2P run-time environment • A distributed persistent storage layer	• A population of interacting agents/ apps • A distributed evolutionary environment • A dynamic, adaptive, learning, and scale-free network infrastructure

Cooperation and competition

To ensure the success of the digital ecosystems agenda as a catalyst of sustainable socio-economic development, it is not sufficient to study and support the formation of communities of SMEs around open-source development activities or the creation of new business models, which form the core of the action within digital ecosystems (in the broad, social science sense). We must also develop a consciously and explicitly reflexive methodology of research that can stimulate and inspire a continuous process of self-renewal and innovation in the SMEs and support their participation in the broader scientific community. One of the fundamental assumptions of the digital ecosystems vision is that such a process is strongly dependent on finding the right balance between cooperation and competition. In particular, whereas competition works well in many market contexts, cooperation can amplify the positive network effects around the formalisation and codification of knowledge within industry sectors, business domains, or geographical regions, for instance in the form of shared business modelling languages and of the open source infrastructure of a service-oriented architecture. How this process works in detail for communities of practice and how it could be leveraged within the digital ecosystems vision, however, is not well understood yet. The working assumption so far has been that combining 1) a greater openness to innovation, 2) a community building process that maximises network effects, and 3) an enabling open source digital ecosystem infrastructure should lead to an environment where knowledge is constantly

[8] http://www.shortcourses.com/how/lenticular/lenticular.htm

created and shared, flowing freely and dynamically where it is needed. This optimistic view has however run into significant practical obstacles that appear to be grounded in deeper philosophical tensions.

Philosophical tensions

As discussed in Dini and Nachira (2007), the ecosystem approach has brought to the fore the deep ontological and epistemological differences between technological research borne out of the philosophical tradition aimed at designing and building "machines" operating in a well-defined, if reductive, objective reality, on the one hand; and social science research, which is more analytical and interpretative ("hermeneutic") in character and aims to account for the interaction between human action and socio-economic and technical structures and processes in the context of policy development, on the other hand. For example, looking at the intersection between social science and Internet technology, it is far from clear whether principles such as decentralised architectures or P2P networks were derived from a particular social theory, or whether instead the converse applies. In general, it seems more accurate to state that socio-economic and technical systems are interdependent and tightly intertwined, that socio-technical and socio-economic phenomena appear to emerge spontaneously from their interaction, and that social theory then tries to explain them. This state of affairs can be interpreted as evidence that it is not so easy to make a clean separation between the "objective" technology we build and our "subjective" or "intersubjective" human experience (Ciborra and Hanseth, 1998).

Fig. 1.1 shows how computer science can play a double role that can be rationalised in a useful practical way from these different points of view, even if they may remain problematic at the philosophical level. From the point of view of social science, the products of computer science are most visible in the form of ICTs whose primary role is to mediate communications between users. From the point of view of natural science, computer science appears to be concerned with the construction of abstract machines whose performance, self-optimisation, and self-healing capabilities could be drastically improved if only we could understand how to emulate biological behaviour in software. By recognising the dual nature of ICT as both communication channel and machine, the figure solves a few problems but leaves a gaping hole: it addresses, even if summarily, software ***use*** and software ***synthesis***, but says nothing at all of software ***design***. Rather than attempting to fit all three aspects of software technology in the same picture, our thread starts again from social science and strikes a new course in the direction of design.

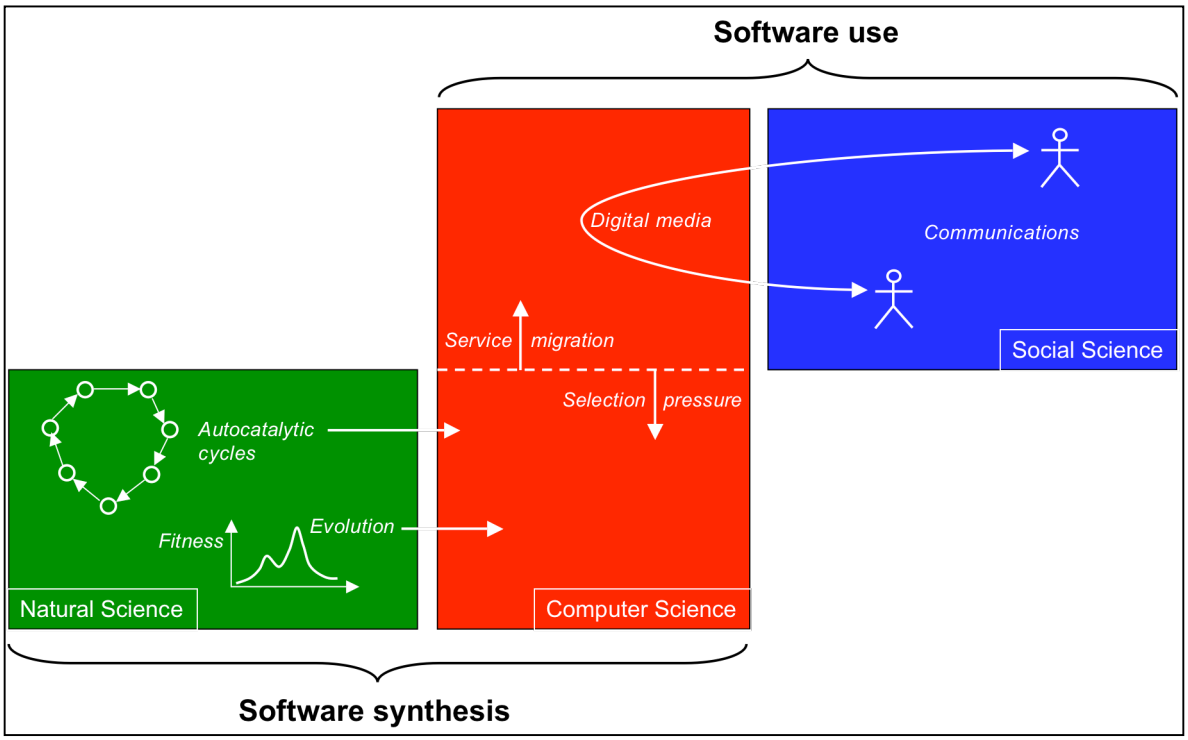


Fig. 2.1 A framework for the peaceful coexistence of the 3 disciplinary domains of DE theory

Language
In digital ecosystems research language is seen as the driver and enabler of the construction of social and economic spaces, ICT as a catalyst of this process, and our role to understand how the constructive power of language can be harnessed by ICTs to realise sustainable socio-economic growth at the regional scale. The importance of language as medium of power relationships (with the attendant challenges in the management of scarce resources and in the governance of democratic institutions) is one of the fundamental assumptions of social constructivism. The introduction of technology into the mix, however, adds another level of complexity.

As discussed in Feenberg (2005), in Heidegger's early writings "Aristotle's conception of being in general is derived from the Greek practice of technical making, from τηχνέ". τηχνέ realises the inherent potentialities of things rather than violating them as does modern technology. Compatibly with this position, according to Marcuse the task of a post-Heideggerian philosophy is to conceive a technology based on respect for nature and incorporating life-affirming values in its very structure, the machines themselves. This utopian demand can be understood as "an implicit recovery of Aristotle's idea of τηχνέ in a modern context, freed from the limitations of ancient Greek thought and available as a basis for a reconstructed modernity". Making things (i.e. engineering) can then be recovered as a life-affirming, deeply human activity, as long as we are not blinded by the myth of the neutrality of technology in an objective world. Feenberg's critical theory of technology shows how technology embodies our cultural values and is in fact an extension of our human languages that necessarily generalises the concept of symbol. The language-technology continuum then contributes to the construction of our understanding of reality and in particular of our social reality.

In this panorama of technology recast as an extension of human cultures and languages ICTs play a unique role because, not only do they share with other kinds of technology this cultural and expressive valence, they *mediate* the very communications that construct the social and cultural systems that created them. It is not clear what the effect of this tight feedback loop might be, but it is pretty clear that it is likely to be a strong one, and perhaps not so easy to control. When looked at through a social science "lens", therefore, the hybrid role of computer science is perhaps best captured by Winograd and Flores' view of computers as communication media (Winograd and Flores, 1987). Because communications, in turn, carry commitments (Austin, 1962; Searle, 1979; Flores and Spinosa, 1998), it becomes easier to accept that ICT has the potential to become a catalyst of social constructivist processes.

The thread that begins with language, therefore, can be seen to account for software ***design*** and software ***use***, but not software ***synthesis*** in the biologically-inspired sense of the previous figure. As argued at the beginning of the paper, software use or more generally ***communications*** do seem to provide an overlap between these very different perspectives. If we examine the finer-grained structure of language we notice that it can be further divided into a more mechanical and objective syntax, and more intersubjective and context-dependent semantics and pragmatics. The levels are in fact many more than two or three (as maintained in intertextual analysis, cultural studies, literary theory, etc). Communications, therefore, appear to span the whole spectrum of media, from machines to poetry. The discussion so far indicates that such a "media stack" is not linear but loops back on itself in a self-reinforcing dynamic. Fig. 1.2 gives an Escher-like graphical rendition of the feedback loops generated by the interaction of ICTs and media content, which could be described through the metaphor of autopoiesis.

Associative and autopoietic systems
Social science and natural science make uncomfortable neighbours. This should not stop us looking for contact points. Fig. 1.1 shows a pragmatic contact point in the ability of computer science to mediate between the two; Fig. 1.2 represents a theoretical contact point in the use of autopoiesis as a metaphor for a self-reinforcing, recursive, and self-generating process that involves only socio-economic and technical systems. A third contact point can be recognised in the resonance between the intersubjectivity of language and the relativist basis of autopoiesis. More generally, digital ecosystems research aims to reconcile the associative (social science) and the autopoietic (natural science) perspectives on socio-economic, technical and natural systems.

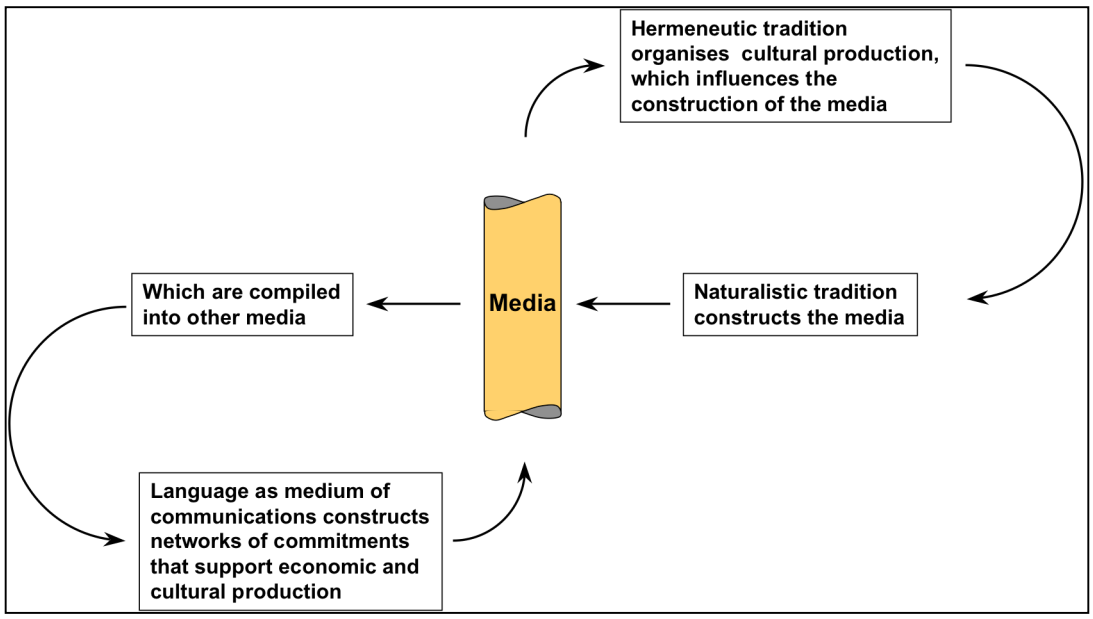


Fig. 2.2 The autopoiesis of media: Where is the boundary between technology and people?

Since Maturana and Varela's first publications on autopoiesis (1980, 1998), this theory has stimulated significant interest in a number of fields such as biology, sociology, law and family therapy. Although this theory has been criticised at least as often as it has been acclaimed, its most appealing characteristic in the context of digital ecosystems research is its strongly relativist position, which makes it stand out among most of the other objectivist theories of natural and physical systems. This is well summarised by Mingers (1995):

> …I think that in a particular respect Maturana's work represents a distinct advance on classical phenomenology, a major criticism of which is that it is essentially individualist and has great difficulty in generating the intersubjective nature of social reality. Here Maturana begins from an intersubjective position. We are (as self-conscious beings) constituted through our language, and language is inevitably an intersubjective phenomenon. As Wittgenstein also argued, there can be no such thing as a private language. Thus language is essentially a consensual domain of agreements, of structural coupling that permits the operations of observers (p. 110).

Even if we acknowledge that autopoiesis has not been able yet to make the transition from a descriptive to an explanatory theory, Minger's words reinforce the impression that it can provide a useful conceptual framework upon which to base a productive dialogue between natural and social science. There remain big methodological differences, but autopoiesis can provide a common epistemological and ontological ground, i.e. how knowledge and a shared reality are constructed through an intersubjective process in both biological and social systems. In particular, the common philosophical problem at the core of a theory of self-organising digital ecosystems, regardless of the disciplinary viewpoint, is how associations and interactions between individual agents or actors can give rise to supra-individual or systemic behaviour, and how global and associative behaviour can in turn influence and constrain—or enable—individual action.

In the physical and biological sciences such interdependence between scales is not in question, although different theories have been developed to account for the observed emergent phenomena in different contexts. In the social sciences, on the other hand, a debate has been raging for centuries around how best to explain and understand social and economic action. Although digital ecosystems research does not pretend to be able to provide final answers to these long-standing questions, it does address challenges of a social, technical, economic, and biological nature through a pluralistic methodology that aims to find a balance between systems and individuals; between context-free models and social processes borne out of diverse cultural and economic contexts; and between optimistic accounts of intelligent and evolutionary

technological infrastructure and qualitative empirical data that documents the conflicts and barriers SMEs face daily to make ends meet.

Fig. 1.3 summarises the systemic interpretation of associative and autopoietic systems and the claim that digital ecosystems can provide an environment of constructive interdisciplinary interaction at the theoretical and applied levels.

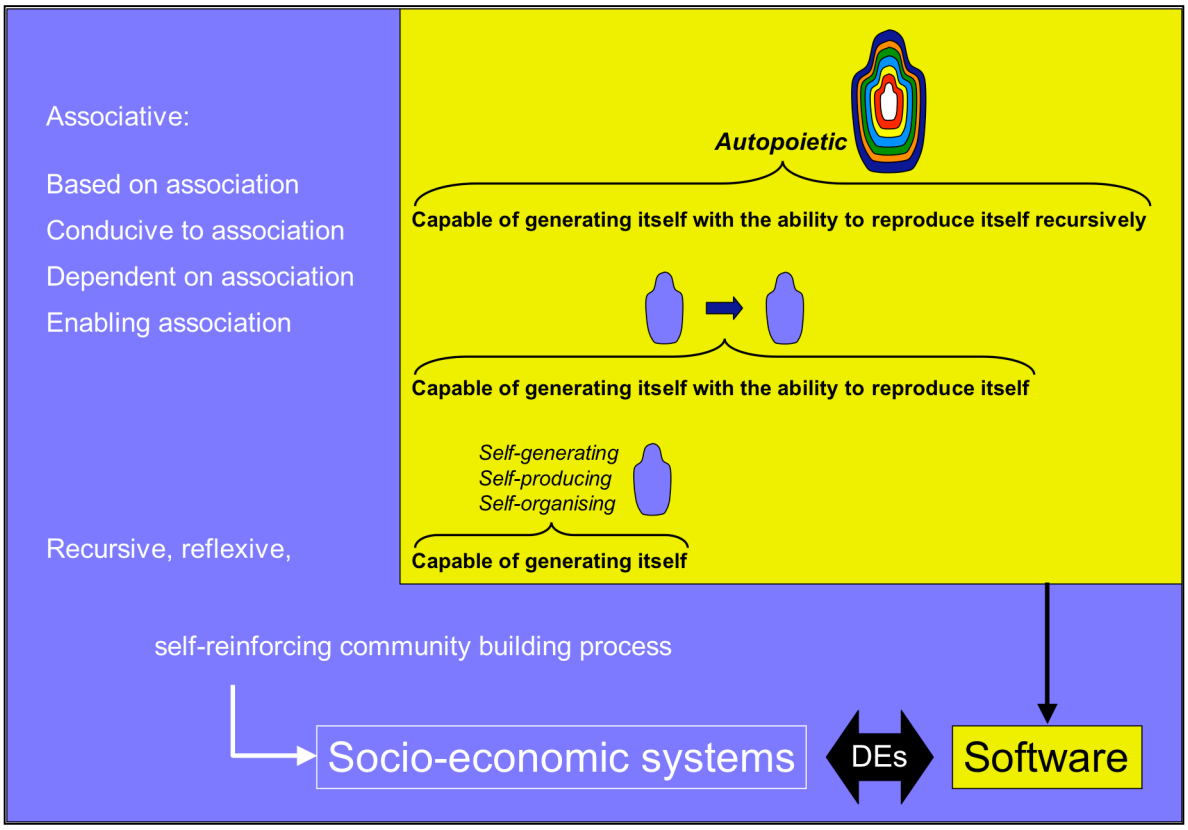


Fig. 2.3 Digital ecosystems enframe socio-economic and biologically-inspired software systems

Computer Science

Fundamental considerations

Touching on some of the philosophical underpinnings of socio-economic and socio-technical systems is uncovering that concepts of power, language, value, and trust play recognisably fundamental roles, based on which the development of a theoretical framework for digital ecosystems is beginning to appear as a plausible possibility from the point of view of social science. Correspondingly fundamental concepts in computer science have been slower in coming, undoubtedly due to the relatively young history of the discipline.

But, what does "fundamental" mean in computer science? There is no limit to the level of abstraction at which data structures or algorithms can be defined in computer science. Everything and anything is fair game. Therefore "fundamental" characteristics or rules of computing systems in a physics or biology sense can only imply the introduction of constraints on the universe of possibilities. How can this possibly be a good thing?[9]

[9] Based on a conversation with Professor Vinny Cahill, Department of Computer Science, Trinity College Dublin (2003).

If we look at computer science as a formal system defined in an abstract and objective space of possibilities, similar to "pure" mathematics and divorced from common human experience, then the argument for leaving it unhindered seems legitimate. If instead we accept computing in its many forms as a vast formal system of languages and technologies that acquires meaning through its interactions with its users, then a relativist epistemology becomes immediately relevant. Whether we wish to treat it as a closed and deterministic system (the "top-down" software engineering process: requirements, specification, modelling, implementation, testing, iterate) or as an open and emergent system (the increasingly popular evolutionary computing "paradigm": mutate, cross-over, implement, select, iterate), the meaning of the software remains entirely dependent to the existence of its users. Take the users away and you are left with arbitrary binary strings.

Or perhaps not? There is the concept of time, or the clock. There is the concept of interaction. There is the concept of state and of state transition. There is the concept of computation as an arithmetic operation on binary strings. Even if we do not force any particular way (architecture) to connect these concepts, there does seem to be some "fundamental" structure after all, a structure that has much in common with the mathematical structure of the physical universe—which we could allow ourselves to endow with some degree of objectivity. These two views could be reconciled by recognising that the theoretical foundations of software engineering and computer science are different, even if they do overlap on many points. In addition, software engineering is strongly biased toward the functional requirements of software applications, whereas computer science is more concerned with the non-functional requirements. We do not pretend to resolve such a complex set of issues for the huge and growing discipline of computing so easily, but Fig. 2.4 proposes such an oversimplification that could serve as a starting point for discussion.

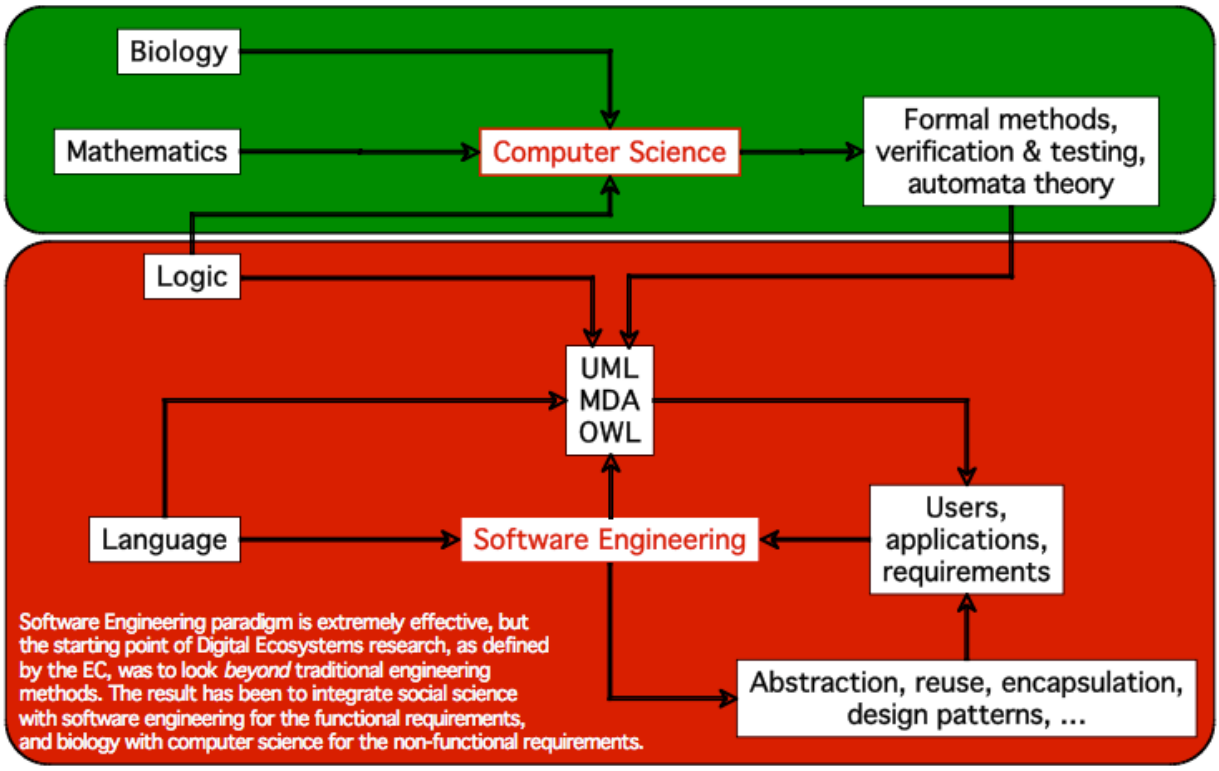


Fig. 2.4 Looking for a useful framework

Assuming that the above is a plausible initial approximation at a framework that can reconcile the different tensions experienced by software engineering and computer science, and by the corresponding communities of practitioners, this figure shows how the work of WP18 needs to focus on mathematics, biology, and automata theory before it can usefully address software engineering concerns. The rest of this deliverable is structured according to this view.

Beyond Turing Machines

As discussed by Golding and Wegner (2005), computer science has in fact been constrained for a long time, specifically since the emergence of the mathematical perspective of computing that identifies algorithms with the evaluation of mathematical functions and that culminates with the Turing Machine "dogma". In the mid-60s Milner started to realise that deterministic finite automata (DFAs) were not quite adequate to model interaction between processes, and he started wondering why automata theory did not allow automata to interact.[10] Perhaps something closer to Mealy automata, which generate an output for every state change triggered by an input, could be harnessed? Fifteen years later Milner and his collaborators (in parallel with Hoare who developed CSP, the calculus of Communicating Sequential Processes) had developed the Calculus of Communicating Systems, which models processes exchanging data along channels that connect them in a fixed topology. Ten years after that he published the π-calculus (Milner, 1999), in which also channel names can be passed along channels, thereby achieving the ability to model variable topology, which is equivalent to mobility.

> Hoare, Milner and others have long realized that TMs do not model all of computation. However, when their theory of concurrent computation was first developed in the late '70s, it was premature to openly challenge TMs as a complete model of computation. Concurrency theory positions interaction as orthogonal to computation, rather than a part of it. By separating interaction from computation, the question whether the models for CCS and the *p*-calculus went beyond Turing machines and algorithms was avoided. (Golding and Wegner, 2005)

Fig. 2.5 shows graphically where interaction fits relative to algorithms and concurrency. Golding and Wegner's paper represents the culmination of a 10-year research effort initiated by Peter Wegner where he addresses the foundations of computer science in order to develop a self-consistent theoretical framework for distributed and interactive computing. He shows how Turing Machines (TMs) were never intended to provide a model of computation for distributed and interactive computing, but were ascribed that role through a series of conceptual adjustments (misinterpretations) of the original theory motivated by practical concerns. This has led to a divergence between computer science and software engineering. Recognising the legitimacy of an extension of the concept of computation and exposing the historical reasons due to which the misconceptions developed, Golding and Wegner provide a clear explanation of the limits of TMs and lay the groundwork for the development of a theory of interaction machines.

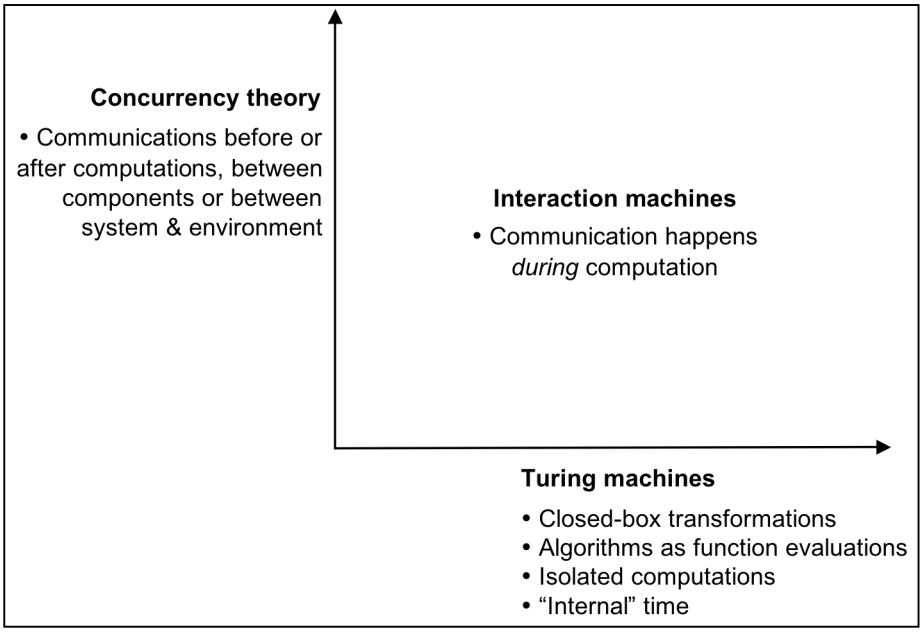


Fig. 2.5 The three main models of computation

[10] Interview with Robin Milner conducted by Martin Berger (2003). http://www.dcs.qmul.ac.uk/~martinb/interviews/milner/

Indeed, Turing himself was very clear about the scope of his formal result. In his 1936 paper (Turing, 1936), he spoke of different kinds of computation, including interactive computation. The technological limits of the first computers that were built in the 1940s and 50s, however, highlighted the single-processor von Neumann architecture and hid the other possibilities. As Golding and Wegner explain, computer science textbooks from the 60s were not uniformly precise in their definitions of algorithm, facilitating the emergence of the perception that all computable problems can be described by an algorithm and that TMs can model any computation. This was partly motivated by the need for software and telecommunications engineering to address practical problems that initially led and soon outstripped the much slower theoretical developments of computer science. They conclude their article recounting the claims that have emerged since the 1936 Church-Turing thesis was published and that make up its common but faulty intepretation. They then provide amended versions of these claims to reflect the original intended meaning of the Church-Turing thesis (Golding and Wegner, 2005):

Claim 1. (Mathematical worldview) All computable problems are function-based.
Claim 2. (Focus on algorithms) All computable problems can be described by an algorithm.
Claim 3. (Practical approach) Algorithms are what computers do.
Claim 4. (Nature of computers) TMs serve as a general model for computers.
Claim 5. (Universality corollary) TMs can simulate any computer.

Corrected Claim 1. All algorithmic problems are function-based.
Corrected Claim 2. All function-based problems can be described by an algorithm.
Corrected Claim 3. Algorithms are what early computers used to do.
Corrected Claim 4. TMs serve as a general model for early computers.
Corrected Claim 5. TMs can simulate any algorithmic computing device.

Furthermore, the following claim is also correct:
Claim 6: TMs cannot compute all problems, nor can they do everything real computers can do.

The view that TMs cannot cope with interactive computing continues to be categorically rejected by most practicing computer scientists. Clearly any algorithm, even in the context of interactive computing, can be represented *a posteriori* by a Turing Machine. The problem arises when one does not know who or what might interrupt an algorithm through an unpredictable and independent external input, when that might happen, and what an appropriate response might be. It is true that we design and implement applications that wait for events before proceeding, or that easily handle being interrupted by unpredictable events, but achieving an appropriate response that has not in some way been programmed in advance seems beyond the range of present possibilities. Whether or not such scenarios are computable by Turing Machines may take a long time to prove in a definitive way to everyone's satisfaction. In the meantime it is probably a matter of personal choice which point of view one chooses to adopt, since such viewpoints do not prevent us from *building* new conceptions and new architectures of software systems. This discussion is still relevant, however, because we need to develop a rationale for attempting a radically different approach at computer science and software engineering. In the absence of mathematical or logical proofs we are limited to relying on "circumstantial evidence" and intuition. If we achieve something useful or interesting we will leave the statement and proof of any corresponding theorems to a later phase of the research.

In Digital Ecosystems research we are examining a further conceptual extension that can be referred to generally as a "distributed algorithm", a term that is increasingly gaining currency in the literature (Babaoglou et al., 2006). In the present discussion we mean an algorithm in a particular software module that is incomplete and can only be completed through interaction with another module or through its "environment". The motivation for such a concept arises from the increasing need for software to adapt automatically to its context, and to continue adapting as its context changes. An architecture capable of handling such a generic and ill-defined requirement differentiates between repeatable, general, and reusable properties of the software and customisable, context-specific properties. Software engineering

has evolved precisely in this direction, through its reliance on design patterns for the former and parametric or case-based adaptation for the latter. Rendering the adaptation process autonomous, i.e. not internally pre-programmed, necessarily requires the software to acquire relevant information externally, from its environment. The only way such information can be aquired is through some kind of interaction.

As the number, scope, and recursion levels of such hypothetical algorithm construction steps through interactions increase, it becomes increasingly difficult to see an infinite linear tape and tape head as an adequate conceptual model of the corresponding computation. Be that as it may, while the search for the most appropriate theoretical model of computation continues it will hopefully not seem too implausible to claim that interactions are as fundamental to the brand of computer science that underpins digital ecosystems in the narrow sense as power, language, value, and trust underpin digital ecosystems in the broad sense.

Do we need more structure?
The introduction of interactions has not constrained computer science, it has actually enlarged it. This is easily seen by the fact that the π-calculus (which is Turing-complete) can represent any computation expressed in λ-calculus (the Turing-complete theoretical archetype of all functional programming languages), but the converse is not true, i.e. not every π-calculus computation can be represented in λ-calculus. Such a strong claim is actually not easy to find stated so starkly in the literature. The sense given here, however, can be extracted from a few quotations taken from the current reference text on π-calculus (Sangiorgi and Walker, 2001), written by two close collaborators of Milner's:

> …the π-calculus can actually do the old job which the λ-calculus does in underpinning conventional programming (R Milner, in the Foreword) … The π-calculus has two aspects. First, it is a theory of mobile systems. … Second, the π-calculus is a general model of computation, which takes interaction as primitive (p. 3). … The λ-calculus, in its untyped or typed versions, cannot describe functions whose algorithmic definition requires that some arguments be run in parallel. … In contrast, the π-calculus naturally describes parallel computations. The π-calculus 'world' is that of processes, rather than functions. Since functions can be seen as special kinds of processes, parallel functions like Por ['parallel or'] can be described in the π-calculus (p. 427). … the class of π-calculus contexts is much richer than the class of λ-calculus contexts … In the π-calculus one can express parallelism and non-determinism, which … are not expressible in the λ-calculus (p. 480).

Faced with the ability to represent and describe the behaviour of an arbitrary number of concurrent processes interacting in arbitrarily complex ways, the response of the formal testing, verification, and simulation methods of computer science (including the π-calculus creators) has been to limit the space of possibilities to formal systems that could be completely defined (the actual number of states for such systems might still be combinatorially large). It is not clear whether we have explicitly limited the expressive power of the calculi of concurrent systems (also known as process algebras) to deterministic systems or whether such calculi are intrinsically limited in this manner. A broader question that seems "fundamental" and worth asking is whether formal systems can be emergent. A linguist would probably reply "Yes". The related more applied question that lies at the heart of digital ecosystems theory (in the narrow sense) is whether computable, or computing, systems can be emergent.

The relevance of emergent structure and behaviour in software is justified by the need to develop software frameworks that can support the adaptation to new and unforeseeable circumstances and requirements. Given the present state of the art, if we relax the deterministic constraints in the formal system that is supposed to model such a software framework, we are left with a ship without a rudder. We are hard-pressed imagining how an incomplete specification of a software system can result into a useful and functioning piece of technology, or how an incomplete formal model can verify anything at all. In such conditions the need for more structure appears evident. The response from the connectionist "camp" of Artificial Intelligence has been the incremental and iterative construction of structure based on the information provided by the target behaviour. Incremental construction implies the presence of a memory mechanism to store intermediate representations, thus making it increasingly difficult to ignore the relevance of biological systems to the discussion.

Whereas on the short time-scale of the individual organism memory is associated with learning, biological evolution generalises the concept of memory to act across generations and over time-scales of the same order as the age of the Earth. In both cases the emergence of structure happens through a passive order construction process that we could equate to a form of pattern replication. In biological systems the pattern is given by sensory inputs or by the selection pressure on a given species arising from the ecosystem within which that species is living. In computer science much simpler frameworks have been developed in the form of neural networks and genetic algorithms, both of which blindly reproduce a desired behaviour or meet a particular set of requirements through many iterations and incremental adjustments that coarsely reflect, respectively, our current limited understanding of neural and evolutionary mechanisms.

This cannot be the final answer. If we assume no relationship between external behaviour and internal structure, then desired external behaviour can only be achieved by a random trial-and-error process of incremental structure formation and rearrangement coupled with a notion of "fitness" and a selection mechanism until the structure finally exhibits the desired target behaviour. If, however, we assume that desired external behaviour is related in some way to internal structure, we might start thinking of other questions. If for example we notice that internal structure is more often than not modular and nested, then we might wonder whether external behaviour might not be somehow related to internal interactions between modules at different scales and between different scales.

Granted that this can be seen as nothing more than the description of any one of the hundreds of thousands of software applications running on the millions of processors and computers in the world today. But how were such applications constructed? Does the fact that they were constructed by relying on the logical relationships between representations at different levels of abstraction, otherwise known as "design", mean that design is the only way to achieve such structures and behaviours? The widespread use of design patterns in software engineering has not led us to wonder if there might be underlying "laws" that give rise to similar patterns in similar situations. We appear to be content to apply the same logical deduction process every time we design an application, a process that starts with user requirements and ends with a class diagram. The class diagram together with other UML views can be seen as a modularisation of code with well-defined interfaces between the modules, so that the writing of functional code within these boundaries can again be deduced logically from the "boundary conditions" themselves. Reuse of patterns cuts down drastically on design time, which is good. But where do these patterns come from?

In physics we can explain the shape of a soap bubble through the local relationships between individual soap and water molecules; but we can also invoke the deeper principle of minimisation of potential energy to arrive at the same geometry with less effort. In the design of complex mechanisms we can likewise invoke global principles and conservation laws that greatly simplify the process relative to what a deterministic Newtonian approach could afford. The reliance on a relatively small number of "fundamental"[11] principles in physics to explain a practically infinite number of observable phenomena is compatible with causal logic over a significant range of length scales, but not over *all* length and time scales. For example, conservation of energy (if we allow for the mass-energy of relativity) applies everywhere and at every scale, whereas deterministic Newtonian mechanics is rather more limited.

Is logic the only framework we can rely upon to construct order? Are there alternative organising principles and frameworks? Logical frameworks are being continually developed, extended, and refined, with corresponding improvements in their expressive power. They help us describe and verify the behaviour of systems defined through formal specifications. This is impressive, but is it enough to

[11] Richard Feynman, one of the foremost physicists of the 20th Century, says in (Feynman, 1965), "I think I can safely say that nobody understands Quantum Mechanics", and in Volume 1 of (Feyman et al., 1965b), " It is important to realise that in physics today we do not know what energy *is* (p. 4-2)" [emphasis in the original]. In other words, more often than not the fundamental concepts of physics remain slippery and mysterious axioms or tautologies, even if they can in fact explain a large number of observable phenomena.

achieve open, adaptive, and emergent software systems? Can we formalise, or specify, useful design patterns within a framework of non-deterministic and emergent interactive computing? The answer to the question at the top of this section appears to be that we need less structure than that afforded by deterministic formal frameworks, but more structure than what evolutionary computation embodies. A mathematical theory that might do just that is the theory of groups, which can equivalently be regarded as a theory of symmetry (Armstrong, 1988) and which lies at the foundations of physics and of much of mathematics.

If we have succeeded in arguing for the need for a theory of emergent computation based on interactions and symmetries, perhaps it may seem justified to bring in biology at this point.

Natural Science

Biological development as metaphor for the construction of order[12]

In 1993 Stuart Kauffman, a theoretical biologist, argued in a long and complex book that in his opinion evolution by natural selection was not a sufficiently powerful order construction process (Kauffman, 1993):

> … Darwin's answer to the sources of the order we see all around us is overwhelmingly an appeal to a single singular force: natural selection. It is this single-force view which I believe to be inadequate, for it fails to notice, fails to stress, fails to incorporate the possibility that simple and complex systems exhibit order spontaneously (p. XIII, Preface).

In other words, something more has been at play to explain the infinite variety of life forms that we observe around us and that are organised in a recursive nested hierarchy of form and function over 10 spatial orders of magnitute from the periodic table to the Blue Whale. The growing general consensus is that morphogenesis and gene expression embody additional "machinery" that adds a "turbo-charger" to the evolutionary order construction process. For example, in a recent collection of papers exploring the boundary between biology and computer science Kumar and Bentley (2003) say,

> Natural evolution has evolved countless organisms of varied morphologies. But before this astonishing diversity of life could evolve, evolution had to create the process of development. … Central to development is construction and self-organisation (p 2) … The main goal of developmental and evolutionary biologists is to understand construction (p 9) … In technological fields, the dream of complex technology that can design itself requires a new way of thinking. We cannot go on building knowledge-rich systems where human designers dictate what should and should not be possible. Instead, we need systems capable of building up complexity from a set of low-level components. Such systems need to be able to learn and adapt in order to discover the most effective ways of assembling components into novel solutions. And this is exactly what developmental processes in biology do, to great effect. (p 10).

Kumar and Bentley go on to ask why we should bother with development when evolutionary algorithms (EAs) can evolve solutions to our problems. The answer lies in the fact that

> …for traditional EAs … typically there is a one-to-one relationship between the genotype and the correspoding solution description. … As solutions become more complex, the length of the genome encoding the solution typically increases. … Instead, evolution by natural selection evolved a highly intricate, non-linear method of mapping genotype to phenotype: development. … Development has enabled evolution to learn how to create complexity. (p 10)

Even though genetic programming operates at a higher level of abstraction, and is therefore correspondingly more powerful in terms of expression of functionality and closer to a non-linear mapping, it relies on the same genetic operators (mutation, cross-over and selection) as genetic algorithms as the basis of the order construction process.

[12] See Dini (2006) for a more extensive discussion.

In fact, evolution is not very creative at all. It relies on sexual reproduction and on mutation to recombine existing traits and to create new traits, respectively. This is the constructive part. Natural selection is the "subtractive" part, i.e. most mutations perish, only the best-adapted survive and reproduce. We are looking for additional principle(s) that can bolster the constructive part. As mentioned above, a potential candidate is the field of mathematical symmetries. Within biology (as a discipline) symmetry has been pioneered as an organising concept by D'Arcy Thompson (1992). How do we make symmetry understandable and, more importantly, how do we connect it to software in a useful way? Let's contextualise the discussion to digital ecosystems before developing this question further.

Whether we call it "non-linear mapping", "distributed algorithm", or some kind of generalised run-time context sensing, it seems pretty clear that we are still looking for an autonomous and stable process framework that is able to construct complex software structures and behaviours from high-level specifications, in response to run-time needs and to specific and information-rich contexts. Fig. 2.6 shows the most recent understanding[13] of how the concepts discussed here may be brought together to provide a starting point for the challenging problem of "software development". We should emphasize that the ideas represented in this figure are still in flux, as we have not actually "solved" the gene expression that we are trying to pose here.

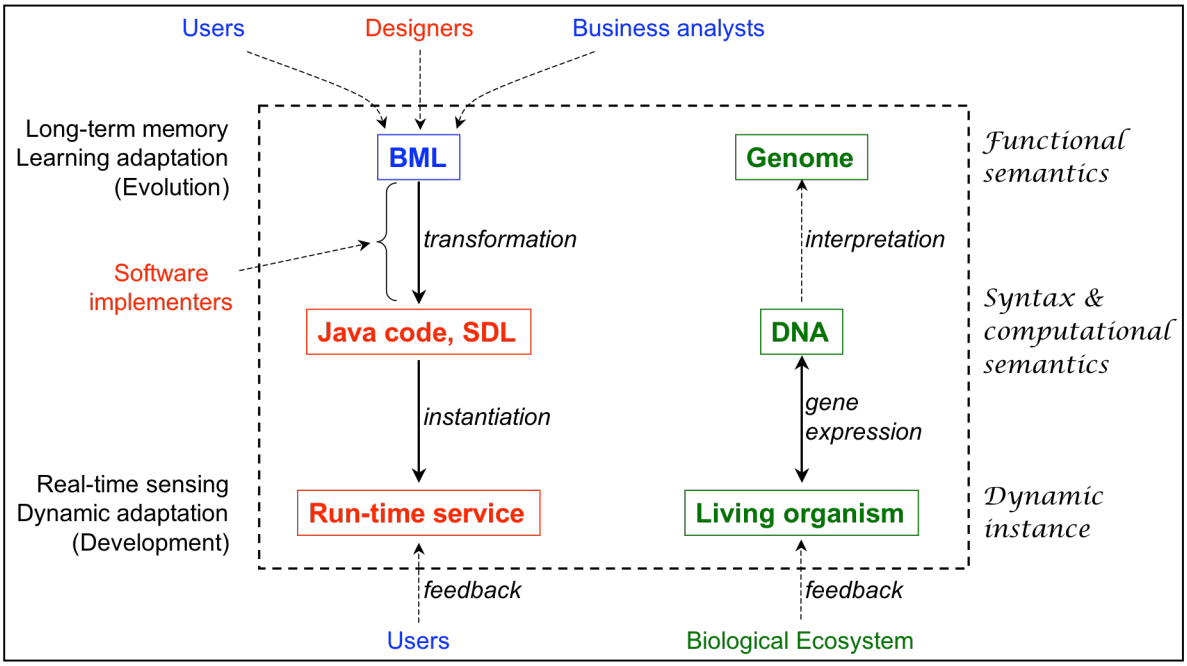


Fig. 2.6 High-level analogy between software engineering and gene expression workflows

DNA can be considered a high-level specification of the organism, and expression of the DNA into a living organism is fundamentally dependent on the interactions between the DNA and its environment. In addition to carrying information on morphological and behavioural traits between generations (evolution), and in addition to serving as the blueprint of the organism (morphogenesis), the DNA also "runs" the body metabolism. Gene expression covers the latter two functions. The interactive character of gene expression and the complementarity of DNA and its environment imply that biological processes are characterised by a distributed, parallel, and collaborative form of "computing", where by computing we mean the breathtakingly fast and hierarchically nested succession of interconnected states and signals from molecular to macroscopic scales.

[13] I am grateful to Gerard Briscoe and Giulio Marcon for their contributions to this diagram.

Whereas gene expression could be associated with automatic code generation (here labelled "transformation"), it seems to map better to run-time instantiation or just-in-time execution. The implication is that BML (Business Modelling Language) models are not anymore regarded as equivalent to "the DNA of the services". This is actually not as big a shift as it might appear at first. In fact, a common interpretation of DNA lumps together its syntactical aspect (code) with its *functional* semantics whereas, strictly speaking, in biology DNA is just a pattern of organic molecules obeying a precise syntax and *computational* semantics.[14] The functional semantics (which gene is correlated to which behavioural or structural trait) are arbitrarily imposed by us and are collectively referred to as the genome of the individual (or the species).

Another point this figure highlights is that whereas in biology species evolve by balancing development with natural selection by the ecosystem, in the context of digital ecosystems we have an opportunity, and a requirement, to enable the "design" or at least the specification of the characteristics of the digital species. The two opposing streams of information that this implies (design-time formalisation and run-time feedback) were recognised from the very beginning of this research. Also in the figure above they can be reconciled by emphasising their different granularities and the adaptive nature of the bottom-up process.

The left column is more problematic. Even if we assume that the UML or BML functional specification can be translated into Java automatically, it is not necessarily obvious or correct to equate Java to DNA. None-the-less, the instantiation of a software entity into RAM does have something in common with gene expression and morphogenesis (although it happens much more quickly), for example in the sensing of available RAM, necessary APIs, and so forth. So the stage between BML and Run-Time Service is labelled as "code" only for the sake of clarity. Our expectation is that in the long term the meaning of "code" and of "code generation" will change. With the continued fragmentation of monolithic applications into declarative/structural code and reusable functional modules, as basic functionality is encapsulated into ever-smaller reusable modules and atomic components, and as these modules are increasingly "shed" by applications and are absorbed by the distributed computational environment, we are not far from being able to jump from BML directly to run-time behaviour.

In the remainder of this chapter we will analyse in greater depth the relationships between the boxes shown in Fig. 2.5, in an attempt ot develop a theoretical framework for the science that is emerging at the boundary between biology and computer science.

From metaphors to isomorphic models

The flip-side of biologically-inspired computing is the new field of computational biology. We should be careful to differentiate between bioinformatics, which is concerned with all of the computer science and software engineering tools, frameworks, algorithms, architectures, and data structures that *support* biological research (for instance the Genome Project), and computational biology, which seeks to utilise computer science concepts and frameworks to *model* biological systems, in particular the biochemistry of the cell. In digital ecosystems research we are concerned with the latter and not at all with the former, other than as one of the many manifestations of "knowledge" that can be mediated by and shared through digital ecosystems and that is therefore relevant to the first half of this article. The convergence between biocomputing and computational biology is very exciting because it is making it possible for biologists and computer scientists to work on the same modelling frameworks even if they are then applied to radically different systems.

Fig. 2.7 is an attempt at an integrating conceptual framework between these two increasingly complementary and interdependent sub-disciplines. The figure relies on the familiar metaphor of the

[14] I made up the terms "functional semantics" and "computational semantics", which in the present context seem to make sense. I believe the latter is actually called "operational semantics", but the mapping is not immediate. For example, Wynskel (1993) breaks down formal semantics into operational semantics, denotational (or mathematical) semantics, and axiomatic semantics. All three categories are much closer to the computational machine than to the "business semantics" or "functional semantics" discussed here. See D18.4 and other DBE deliverables (e.g. from WP9) for further discussion.

organism (blue rounded box) embedded in an environment or ecosystem (green background), implying an analogy with a software service embedded in a digital ecosystem (in the narrow sense). The text-boxes shown, however, are not the components of such an organism/service or ecosystem/digital ecosystem. Rather, they are those areas of research in computer science, biology, and mathematics that are relevant to the different parts of the metaphor and that have already been accepted as central (most of the lower half of the figure), that are gaining acceptance (top-left quadrant), or that are far from proven and are currently being assessed in digital ecosystems research (all the references to symmetries).

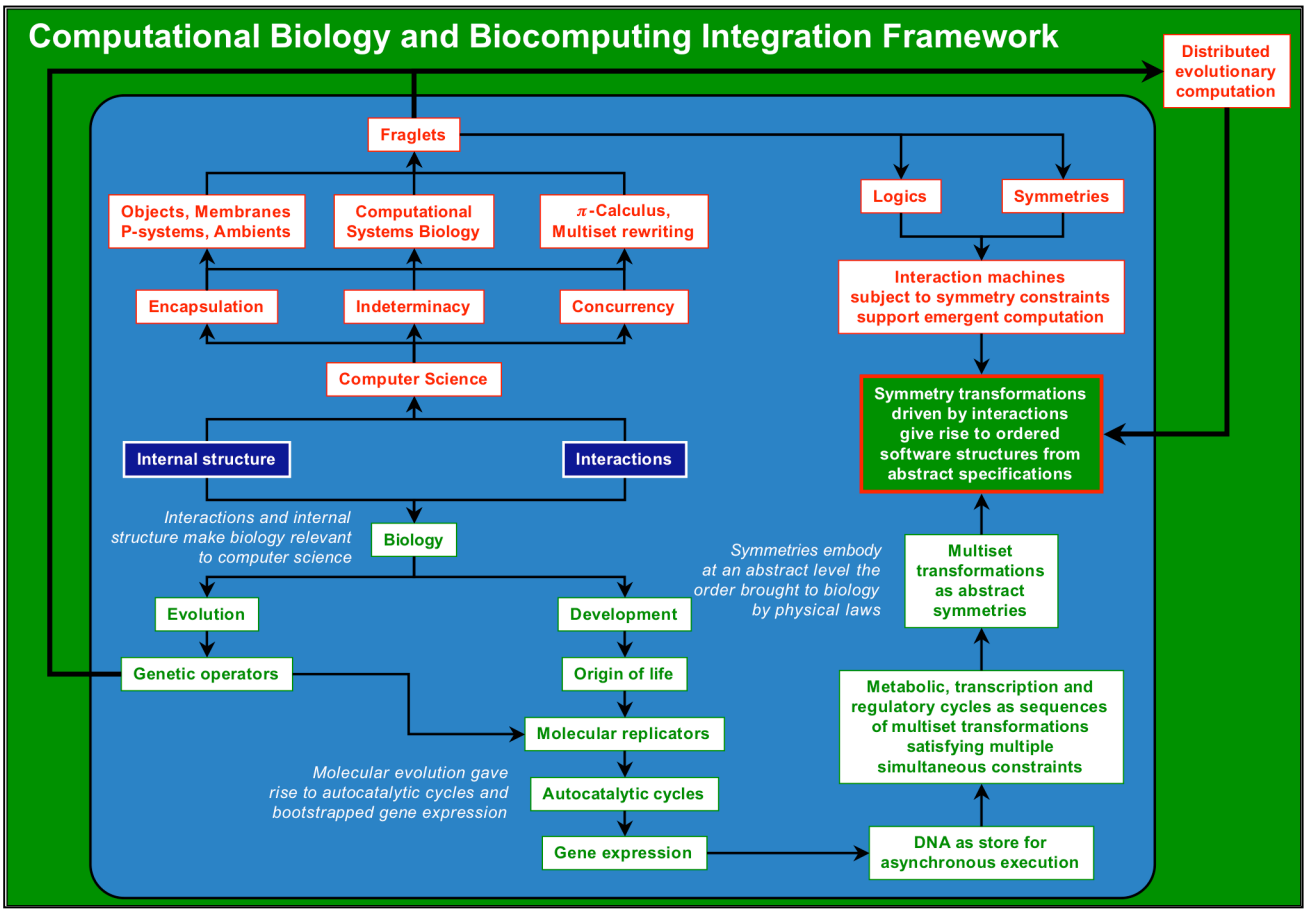


Fig. 2.7 Toward a common mathematical framework for biology and computer science

The figure starts with the concepts of interactions and internal structure and presents those aspects of computer science that, by chance or by design, are giving rise to an increasingly "biological" conception of computing. Digital ecosystems research so far has mainly been concerned with the mapping of all the boxes shown on the left, associated with biology and computer science, and with the development of a distributed architecture that can support evolutionary computing, shown on the outside as a feedback. The same concepts of interactions and internal structure are central also for a rationalisation of biological systems through the processes of evolution and development, whose interdependence has led to increasingly complex organisms and processes, such as gene expression. It is helpful to recount very briefly how this might have happened.

Evolution has evolved gene expression. Thus, evolution is indeed the starting point. At some point, by chance, some of the molecular species populating the primordial soup acquired the ability to make copies of themselves. The first replicators were born. The competition between replicator species in the race for the absorption of sunlight and loose nutrients (sugars or whatever else) in some cases turned into cooperation, where two or more interdependent species acquired an evolutionary advantage over species acting on their own. These were the early "value chains" of the primordial soup. In the case where the

chain of interdependencies between replicators closed back on itself an "autocatalytic cycle" formed. This cursory reference could never do justice to the large bibliography on replicator dynamics (Eigen and Schuster, 1977-78; Stadler and Stadler, 2003; Stephan-Otto Attolini, 2005) that, over the past several decades, has pieced together a plausible scenario for how the initial molecular replicators gradually bootstrapped themselves into RNA and slowly into families of interdependent structures. When one such structure, the membrane, closed around such a family in a symbiotic (i.e. mutually beneficial or synergistic) relationship the first cell was born.

The cell is composed of parts that conspired to reproduce themselves at each step of their evolution. It is a super-structure that organises sub-structures, that are themselves nested hierarchies of structures. The behaviour of such structures is also nested and recursive. Thus evolution favoured those replicators whose replicating ability depended upon, and reinforced, the replicating ability of their partners. Furthermore, evolution favoured those replicators whose behaviour enabled the group of replicators to replicate as a unit. This is how autopoiesis came about. A stem cell is the best example we have of an autopoietic system.

In this paper I argue that to reach the chimera of self-organising software systems we need to understand how to model the order construction processes of biology through symmetry groups. In biology symmetries are symptoms of universal physical laws that, by acting upon a stable alphabet of chemical elements, are responsible for all the order construction that we see in the organic and inorganic worlds. Because software systems are abstract and do not rely on underlying physical laws, the role of symmetries as symptoms of underlying order can be inverted to become mathematically formalised constraints that make possible the construction of an equivalent order out of the range of possible structures and behaviours of software. To clarify, in the software engineering methodology called Programming by Contract certain preconditions and post-conditions are specified and enforced when a call is made on a module. These are clearly constraints. In this discussion the constraints we are referring to apply to the software that needs to be created *between* the preconditions and the postconditions, to enable the highest possible level of automatic generation of code (or behaviour, as mentioned above).

The constraints need to act on something, and that something is a large number of interacting components over a wide range of scales. The presence of interactions driven either by the “external” environment (users, other components, network messages, RPCs, etc) or by pre-programmed internal random “mixing” processes is analogous to a non-zero temperature in a physical system. Thus the conceptual basis of this picture is clearly inspired by statistical physics: the global state of the system results from a balance between the tendency toward order that is observed as the temperature approaches zero and that manifests itself in the solid state, as regular and symmetric crystals, and the tendency toward chaos that is observed as the temperature increases without bound and that manifests itself in the gaseous state, as a structureless collection of randomly colliding components. Biological systems, in fact, are able to strike this balance and depend mainly on the liquid state, although they involve all three phases of matter.

Because for pragmatic purposes we need to accelerate the evolutionary dynamics of digital ecosystems and, equivalently, we have a further requirement for the software to reflect directly the needs of the users, we need to meet an additional challenge. Digital ecosystems are immensely complex and many-dimensional, but in a sense they are semi-infinite media since by definition they will always present one interface to the users. Thus the order construction processes and the interaction frameworks discussed here will always need to be compatible with one “boundary condition”: the specification of the services and the behaviour of the users that the software needs to adapt to. Whereas the latter can be usefully harnessed through the sub-symbolic learning approaches of Connectionism, the former points to the need for the interaction-driven and symmetry-based order construction framework outlined here to “grow” out of abstract specifications. In other words, it would appear that we also need to find a connection between group theory and formal logics.

The argument developed in the DBE Science Vision report, already cited, led to the “fundamental dychotomy of the DBE” which, following Varela (Varela et al, 1991), pits symbols against behaviours in

both biology and cognitive science as opposite perspectives on order construction. The discussion in this article could be summarised with the claim that the group-theoretic approach has the potential to reconcile this dichotomy, as shown in Table 2.2.

Table 2.2 The fundamental dichotomy of the DBE: A possible reconciliation?

	Evolutionary Biology	*Group Theory*	*Cognitive Science*
Symbols	GENETIC DETERMINISM[15]	EMERGENT SYSTEMS	COGNITIVISM
Symmetries			
Behaviours	AUTOPOIESIS		CONNECTIONISM

The connections between logic and groups are a current area of research and will not be discussed further in this article. The article will instead review briefly existing modelling frameworks from physics, chemistry and computer science. This will lead to a very innovative model of computation borne out of the practical concerns of communication protocols, the Fraglets. The discussion will end with a simple example of how symmetries can connect biology to computer science.

In search of the right model

The dialogue between the wide range of modelling frameworks that have been developed over the past hundred years in physics, biology, and chemistry and computer science began about twenty years ago. In digital ecosystems research this kind of analysis was begun in Deliverable D18.1 and is on-going.[16] Some such frameworks have tended to be based on assumptions of continuity or on the validity of averaging simply because this affords a huge dimensional reduction in the number of variables needed to be accounted for. Unfortunately, however, such assumptions work well only for systems that are very large (Avogadro's number of particles) or in equilibrium, or both. The cell is much smaller (10^3 – 10^9 particles as opposed to 10^{23}) and can only function by virtue of a flow of energy and nutrients maintaining it *away* from equilibrium. The motor force that drives self-organising processes in fact comes from the cell's continuous "fall" toward equilibrium, which it never reaches as long as the organism remains alive.

Although the non-linear coupling of dynamical systems theory is undoubtedly relevant to this discussion, differential equations do not seem to be "granular" enough. They model the time evolution of average concentrations of reactants and products, thereby making it difficult to resolve the spatial variations within a particular cell compartment (such as the nucleus) or the overall topology of the cell. Reaction-diffusion systems based on partial differential equations do better in resolving spatial variations (by definition), but remain limited in the number of variables they are able to model simultaneously.

The study of non-equilibrium statistical mechanics and critical phenomena could be seen as attempting to strike a compromise between equilibrium statistical methods (applicable to large systems in equilibrium) and dynamical systems theory (applicable to small unstable systems). One of the appealing aspects of such approaches is their ability to model the communication between different scales of the system. However, although phase transitions and critical phenomena are a direct consequence of the presence of interaction forces between the system's components, which sounds close in concept to interactive computing, it is difficult to imagine how an interaction potential energy function of very limited expressiveness could usefully encode arbitrary information exchange between software modules.

In the absence of a direct link between physical interaction forces and order construction processes, we are left with no choice but seek a rationale for self-organisation at higher levels of abstraction. It seems better to focus on the state changes that result from interactions rather than on the modelling of the coupling mechanism itself. A significant amount of work in this direction has come from modelling

[15] See, for instance Dawkins (1989).
[16] See Dini and Berdou (2004).

frameworks inspired from chemistry, generally referred to as multiset rewriting (Banatre and Metayer, 1993; Giavitto et al., 2004; Stanley and Mikkulainen, 2003). Multiset are sets where more than one copy of each element can be present, and "rewriting" refers to the rewriting of chemical reactants into chemical products as a result of a chemical reaction.

A conceptually similar approach has extended the π-calculus into the Stochastic π-Calculus by assigning reaction probabilities between different processes, which then become relevant to the modelling of biochemical systems and have given rise to the field of Computational Systems Biology (Priami, 2005). The stochastic π-calculus has been used with interesting results by Shapiro and co-workers (Regev and Shapiro, 2002). A wealth of detail and some very interesting examples can be found in Regev's PhD thesis (2002). In essence these approaches rely on associating a π-calculus process with a bio-molecule such as a protein or enzyme. Communications between processes become physical interactions or chemical reactions between molecules. Regev shows how this approach can reproduce phenomena such as the cell circadian clock[17] as long as the relevant chemical species and their concentrations are set up correctly at the beginning of the simulation.

Regev also explores a more recent example of computer science-based biochemical modelling, Cardelli's Ambient Calculus (Regev et al, 2004). Ambients, as an alternative approach called P-Systems (Paun and Rozenberg, 2002), are a form of membrane computing. Both approaches are concerned with a model of computation that uses a membrane as a primitive and is therefore implicitly capable of resolving the topology of the biochemical environment of the cell.

As a final point in this extremely brief review it is worth noting that the π-calculus has also served as one of the theoretical reference frameworks in the development of BPEL (Business Process Execution Language) (Havey, 2005). BPEL is the syntactical language being used in the DBE project to specify the interfaces between the service modules a complex service and their order of execution, and therefore underpins service composition.

As we survey the rather wide field of possible modelling approaches that have the potential to achieve the challenge of self-organisation of software, we reach two high-level requirements:

1- from the point of view of biology, the model must be able to reproduce ordered patterns and behaviour of open biochemical systems to an appropriate level of granularity. In addition, the model must be able to resolve the topology of the biochemical environment.
2- from the point of view of computer science, the model must combine expressiveness with the ability to support interactions and real-time non-preprogrammed adaptation to unforeseen events and situations. As a special case, it must also be able to support the generation of a running instance of a service or application based on a high-level symbolic specification of its functional behaviour.

These are tall orders, but a model that, although in its early stages, seems to hold the potential eventually to satisfy these requirements is the Fraglets.

Fraglets: a new way of computing?

It seems fair to say that Fraglets (Tschudin, 2003)[18] represent a radically different approach at computing. Fraglets are based on theoretical computer science frameworks that are centred on the concept of interaction, such as process algebras (π-calculus) and multiset rewriting (artificial chemistry). They are sufficiently versatile to support the encoding of deterministic algorithms, but can just as well support multiple levels of non-determinism, as explained below. They intrinsically embody interactions and can *only* be implemented as a distributed algorithm. Because they were first developed to implement communication protocols they can also very easily model different topologies that can equally well represent network nodes or cell compartments.

[17] Wikipedia: a roughly-24-hour cycle in the physiological processes of living beings
[18] See www.fraglets.org for a copy of this paper and the open source (GPL) downloadable execution environment

Other aspects of Fraglets that are not immediately relevant to this discussion, but that are important for biocomputing in general and digital ecosystems in particular, concern their extension toward genetic programming (Yamamoto and Tschudin, 2005), membranes (i.e. nested containers), Java APIs for atomic function implementation and reuse in engineering contexts, and variable topology (i.e. mobility). Aspects that are relevant to this discussion and that are currently very much the focus of research include a store to model asynchronous DNA function and a prototypical autocatalytic cycle as the atomic or archetypal autopoietic system. These topics are all work in progress or possibilities under evaluation, for instance in the OPAALS and the BIONETS EU projects.[19]

Fraglets are strings of instructions and/or data, or arbitrary length. Their syntax is very simple:

A[keyword1 : keyword2: ...] N

where "A" is the name of the node in which the fraglet is located and "N" is the number of identical fraglets of a particular type being defined at the beginning of the run. Each keyword can be either an instruction or data. Data can be a "payload" that needs to be transmitted between nodes or a token (essentially a pattern) that e.g. can be matched with another fraglet's keyword to allow the reaction between them to proceed. Only the first keyword or the first two keywords are "active" in each fraglet at any one time, which means that execution proceeds by header processing. Each fraglet is thus equivalent to a sequential process. However no fraglet is (usually) self-sufficient. For a meaningful action to be executed several fraglets need to interact to activate the required sequence of instructions. Thus the algorithm that needs to be executed is actually distributed over a set of fraglets.

Fraglets do not just interact, they can also be executed on their own, depending on the instruction being executed. Fraglet semantics can be summarised as only seven instructions, as shown in the table.

Table 2.3 Fraglets instructions (reaction and transformation rules) (Tschudin, 2003)

Reaction	*Input*	*Output*	*Semantics*
match	[match : s : $tail_1$], [s : $tail_2$]	[$tail_1$: $tail_2$]	concatenates 2 fraglets with matching tags
matchp	[matchp : s : $tail_1$], [s : $tail_2$]	[matchp : s : $tail_1$], [$tail_1$: $tail_2$]	Persistent match (works as an enzyme, i.e. it is not consumed)
Transformation			
dup	[dup : t : u : tail]	[t : u : u : tail]	duplicates a symbol
exch	[exch : t : u : v : tail]	[t : v : u : tail]	swaps two symbols
split	[split : t : . . . : * : tail]	[t : . . .], [tail]	breaks fraglet at position *
send	A[send : B : tail]	B[tail] (unreliably)	sends fraglet from A to B
wait	[wait : tail]	[tail] (after interval)	waits a predefined interval
nul	[nul : tail]	[]	fraglet is removed

Fig. 2.8 shows an example of a communication protocol that is explained in detail in Tschudin's 2003 paper already cited. Here we will focus only on a few points relevant to the present discussion. The two coloured boxes represent two nodes of a network. All the fraglets in the yellow box should start with an "A" and all those in the cyan box should start with a "B". In this figure, however, the intent was to emphasise the difference between the initial set of fraglets in each node and the fraglets that are generated at run-time, through interactions and transformations. The initial set is recognisable by the starting letter and boldface type; all the other fraglets are generated at run-time. This example is also helpful for showing how, for example, the yellow node could be the cytoplasm and the cyan node could be the nucleus of the cell. In other words, the same computing model can be used for biocomputing applications and for computational biology simulations.

[19] www.opaals.org (Contract number FP6-034824); www.bionets.org (Contract number FP6-027748)

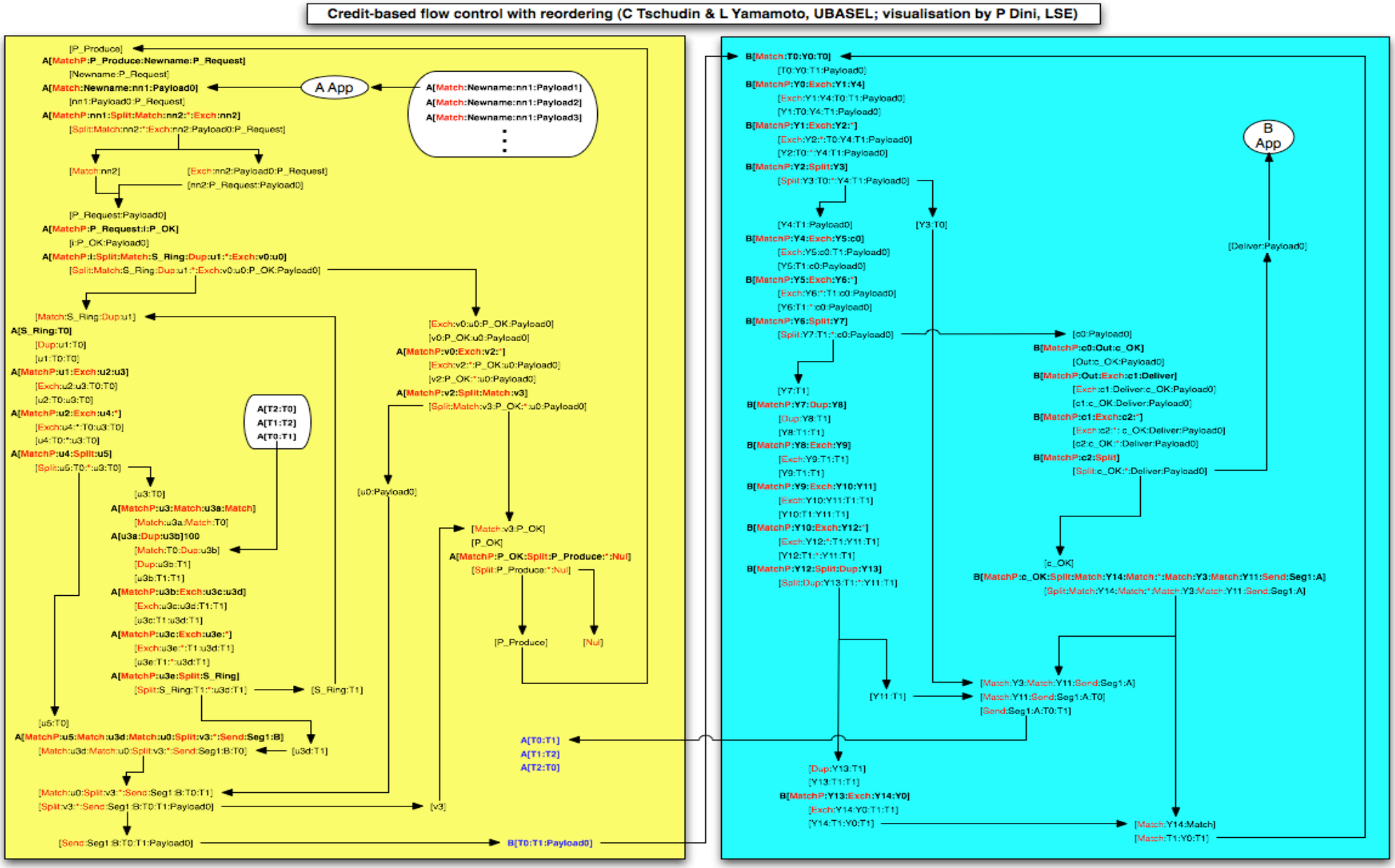


Fig. 2.8 Fraglets example showing the explicit algorithm (regular font) along with the implicit initial fraglet multiset (bold font)

Clearly the initial set makes no sense by itself. When displayed together with the fraglets it generates, on the other hand, an execution path can be recognised. The execution of a multiset of fraglets proceeds by emulation of chemical reactions. In other words, at each time-step the whole set is examined and a fraglet (or two interacting ones) is picked from the subset of actions that are possible at that point. Even though the current Fraglets execution environment uses interleaving to simulate parallelism, the execution could be parallelised over multiple processors without loss of semantics.

This particular example is deterministic in spite of the forks in the execution path, in the sense that it will perform the protocol function that was specified. It may activate or generate intermediate fraglets in a slightly different order each time it is run (if a true random number generator is used), but the effect will be the same. Therefore, this particular example is equivalent to the same protocol written in a standard language such as C. The implication is that the initial set of fraglets could be seen as a "compressed" form of the full algorithm shown in the figure. The forks in the execution path, however, point to the possibility of programming the multiset to do different and mutually exclusive things depending on which path is chosen at a fork. Since at each time-step the next action is chosen randomly from the set of allowable actions, a Fraglets program could be designed to be non-deterministic at one or more forks. This is the first level of non-determinism mentioned above.

The current emphasis of digital ecosystems research in the construction of order, however, is concerned with the next level of non-determinism. In particular, is it possible to devise a multiset of fraglets that is incomplete, i.e. that does *not* correspond to a unique and complete program even when expanded, and that is still capable of executing a useful or meaningful function? Based on the concepts discussed so far in this article the answer is "Maybe". It appears that the missing information would need to be acquired from the environment the Fraglets program is interacting with. And that it might be possible to derive the structuring of such information by relying on symmetry properties of the fraglets. This hypothesis is inspired by the tight interdependence between structure and function in biology. We can say little more at this point about whether this line or enquiry might be successful or not. We close the article with a reductionist look at the connections between symmetries, biology, and computer science.

Toward a simplistic model of digital enzymes

The following facts are surprising to the non-initiated and somewhat mind-boggling (Alberts et al., 2002):

> A typical enzyme will catalyze the reaction of about a thousand substrate molecules every second. … Rapid binding is possible because the motions caused by heat energy are enormously fast at the molecular level. … a large globular protein is constantly tumbling, rotating about its axis about a million times per second. … Since enzymes move more slowly than substrates in cells, we can think of them as sitting still. The rate of encounter of each enzyme with its substrate will depend on the concentration of the substrate molecule. [In a typical case,] the active site on an enzyme molecule will be bombarded by about 500,000 random collisions with the substrate molecule per second (pp 77-78).

When faced with the task of modelling biological processes, upon encountering descriptions such as the above it is difficult not to think of the cell biomolecular system as an immensely powerful digital finite state machine. The cell "computes" in the sense that interactions between molecules change their states, and these state transitions correspond to the execution of some function, just like an algorithm. Such functions are called metabolic, catabolic, transcription, or regulatory cycles, and are themselves components of higher-level functions inside and outside the cell, which could be said to culminate with the mind. What makes the cell machinery so powerful is its parallelism. Even though the 1 KHz rate of interaction of an enzyme with its substrate is very slow compared to the Mac upon which I am writing this article (2.33 GHz dual core), which is 4.66 million times faster, in a typical cell there are many millions such interactions happening in parallel every second, performing hundreds of metabolic cycles simultaneously, thus making each cell significantly more powerful.

Having argued in a hopefully convincing way about the fundamental importance of interactions in biocomputing, we have identified the Fraglets as the most promising model that starts with a "foot" in engineering but that could be expanded in many ways toward biology thanks to its theoretical basis in

process algebras and multisets. Now we are looking at the other side, starting with biology and developing a theoretical framework that is compatible with digital systems. In this article we will not be able to find the meeting point of these two tunnels under the mountain that currently divides the valley of deterministic computing systems from the valley of emergent biological systems. We only mean to show what could be a plausible starting point and direction for the second tunnel.

Because this article is not the place for a diversion into group theory and symmetries, a certain level of familiarity with the basic concepts must unfortunately be assumed. It may help to realise that mathematical symmetries can be seen as generalisations of the more familiar and commonplace concept of symmetry. If a symmetry is defined as a transformation that leaves something invariant, then the common interpretation of the term could be stated as "invariance of shape with respect to 180-degree rotations about the vertical axis". In examining a typical Fraglets program such as the one shown in Fig. 2.7 it does not take long to realise that some patterns of execution tend to repeat themselves when similar tasks are being performed. This recurrence is an example of a simple form of invariance and is one of the observations, along with D'Arcy Thompson's symmetries, that is motivating this line of enquiry.

Because the active site(s) of enzymes occur at discrete angular intervals and because enzymes rotate to look for a match with their substrate, it seems worthwhile to investigate the rotational symmetries of the regular solids.

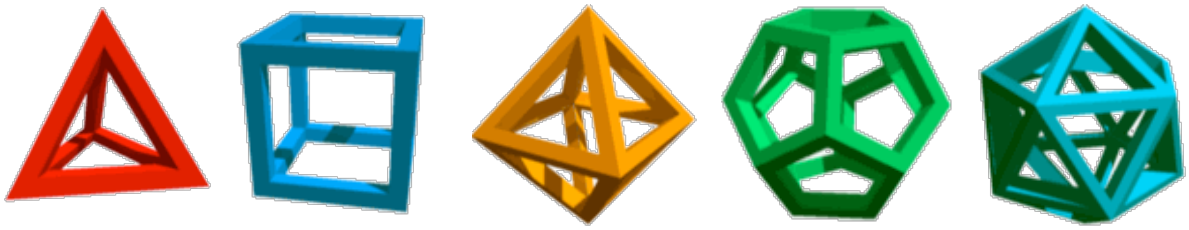


Fig. 2.9 The five Platonic solids[20]

Fig. 2.10 shows the 12 rotational symmetries of the tetrahedron, where "e" stands for the Identity transformation.[21] Interestingly, these can be expressed as combinations of the two rotations "r", 120 deg rotation about the vertical axis, and "s", 180 deg about an axis bisecting two opposite edges. When the tetrahedron is rotated by either r or s its shape relative to a stationary observer does not change. Furthermore, when it is rotated by any combination of r's and s's, its shape will also remain invariant. The figure shows 12 different (but non-unique) combinations of r's and s's that give all 12 symmetries. If these combinations of rotations are viewed as a binary alphabet, it is straightforward to define a deterministic finite automaton (DFA) that will accept any string of r's and s's, Fig. 2.11. Clearly one needs to define the accepting state(s), and depending on which such state(s) is/are chosen a different regular language will be recognised by the automaton. Figs. 2.12 and 2.13 show the equivalent analysis for the cube, which turns out to be isomorphic to the octahedron.

This suggests that a possible strategy to "translate" metabolic behaviour into a stream of digital symbols—and back. The former is conceptually related to a monitoring or analytic function of cell metabolism, whereas the latter is related to a modelling or synthetic function. There are huge challenges still to be overcome, clearly. Furthermore, rotations are suggestive but are only an example of transformations (state transitions) that could be useful or semantically meaningful for biological computation. It is not clear what other kinds of transformations might be useful.

[20] http://math.about.com/od/geometry/ss/platonic.htm
[21] A complete discussion of the symmetries of the Platonic solids can be found in Armstrong (1988).

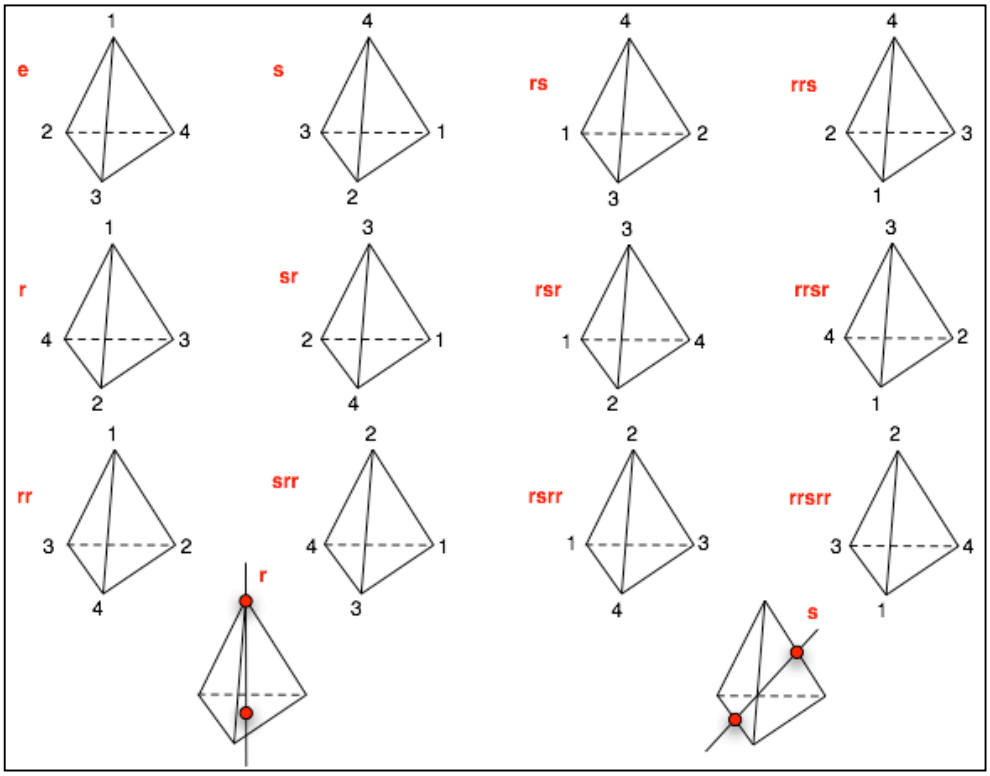


Fig. 2.10 The 12 rotational symmetries of the tetrahedron, generated by 2 rotation axes (Alternating Group A_4)

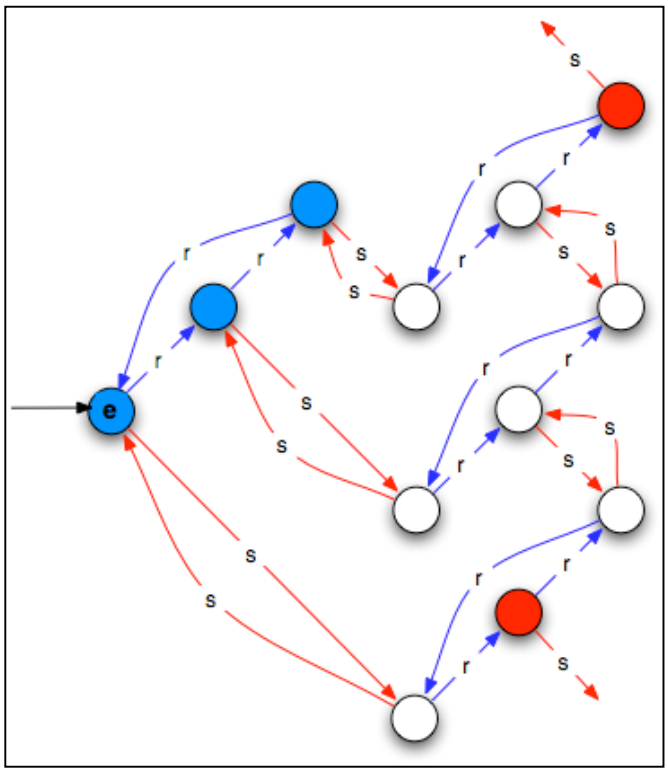


Fig. 2.11 The 12 rotational symmetries of the tetrahedron as the 12 states of a Finite Deterministic Automaton whose language is given by any string of "r" and "s" symbols

If we take the rotations not as absolute but as relative rotations between an enzyme and its substrate, it may be possible to imagine a stream of bits issuing from their interaction. Each interaction, however, may produce a very short stream indeed, since its result is generally the formation of a new compound whose geometry is different. Each time a new compound is generated only a few state transitions are effected. How are the millions of fragmented digital streams being generated in parallel to be integrated into a coherent high-level description of functional behaviour? Interestingly, this is qualitatively not unlike the behaviour of the Fraglets.

It may appear that this approach is too low-level. However, another fundamental aspect of biological systems is their recursive structural and functional organisation. This means that higher-level structure and behaviour is built with nested lower-level structure and behaviour that in many cases replicates the higher-level characteristics at a smaller physical scale and faster time scale. The mathematical objects that model scale invariance are Mandelbrodt's Fractals, which are symmetric in yet another way and that have been used to model countless biological (and physical) structures.

Clearly the Platonic solids are not realistic models of real enzymes. The more important point is to build a modelling framework that replicates, however crudely, the main aspects of the mathematical structure that underpins biological computation. It is entirely possible that some abstraction of the solids, similar to the Fraglets, will be able to achieve the required expressiveness and recursion capability. The Platonic solids in this case could simply serve as an intermediate conceptualisation tool, to aid our intuition in building a bridge from proteins to bit streams, and possibly to help us visualise distributed interactive computation in the future. The next step might therefore be to construct "atomic" DFAs that correspond to Fraglets state transitions and that could be combined into larger finite state machines whilst retaining certain structural characteristics that correspond to particular behaviour.

As a final contextual point it is helpful to remind ourselves of the Chomsky (1957) hierarchy of languages:

- (Type 0) Turing Machines (TMs) are the most general and recognise anything (although not necessarily in finite time)
- (Type 1) Bounded-tape Turing Machines are more complex and recognise context-sensitive languages
- (Type 2) Pushdown Automata (PDA) have a stack and recognise context-free languages
- (Type 3) Deterministic Finite Automata (DFA) recognise Regular Languages (simple commands like "ls" in Unix to list directory files)

Java falls somewhere between Type 2 and Type 1, but is closer to Type 2. The extremely simple examples we are discussing here belong to the lowest level of the hierarchy (Type 3). There is no proof yet that Fraglets are Turing-complete (Type 0) but it is suspected that they are. If what we have argued in this article is correct, the credit-based flow control with reordering protocol of Fig. 2.8 would be an example of a deterministic algorithm not unlike Java. The interesting fundamental and still open question is whether by relaxing the determinism of Fraglets programs in the manner suggested here we might be climbing the hierarchy to the top, perhaps going beyond Turing Machines themselves.

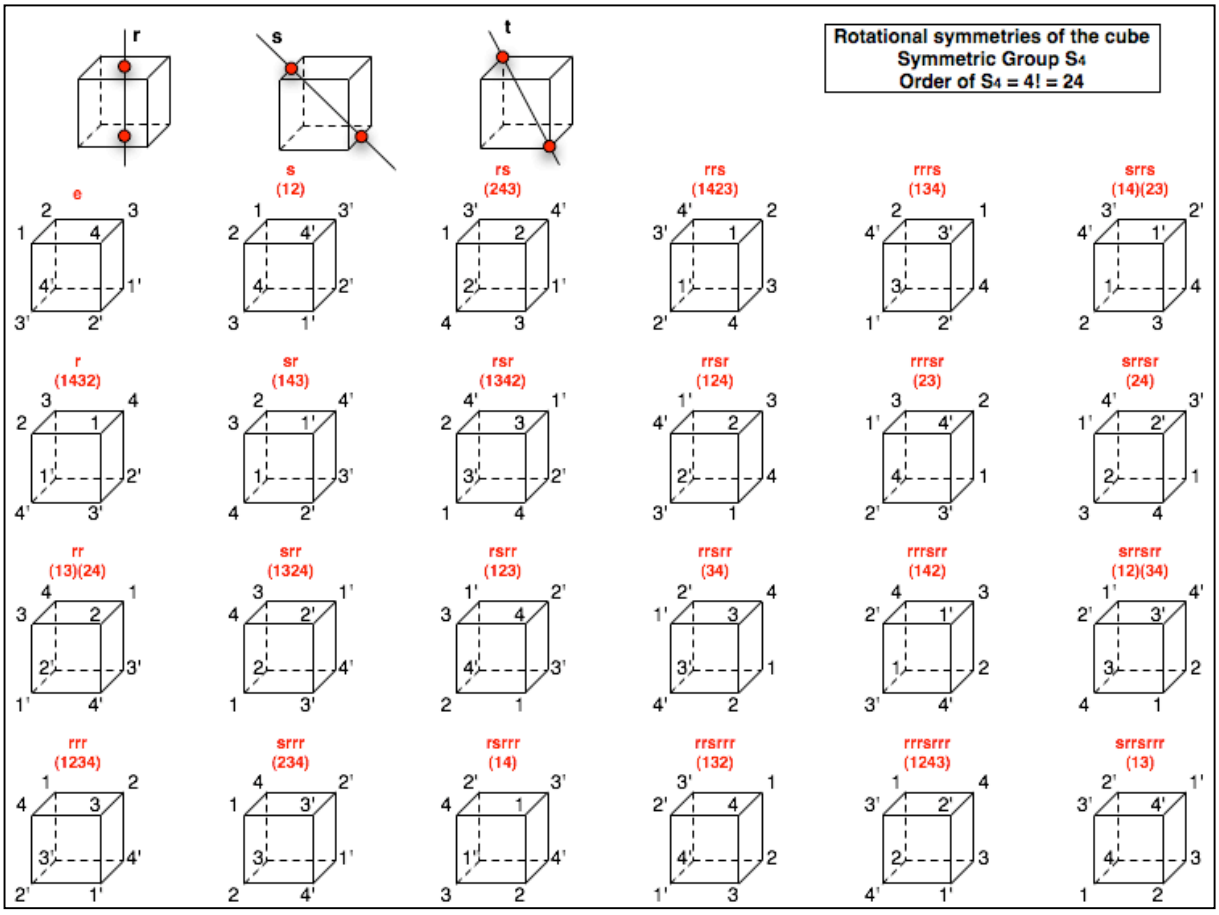


Fig. 2.12 The 24 rotational symmetries of the cube, generated by 2 rotation axes (Symmetric Group S_4)

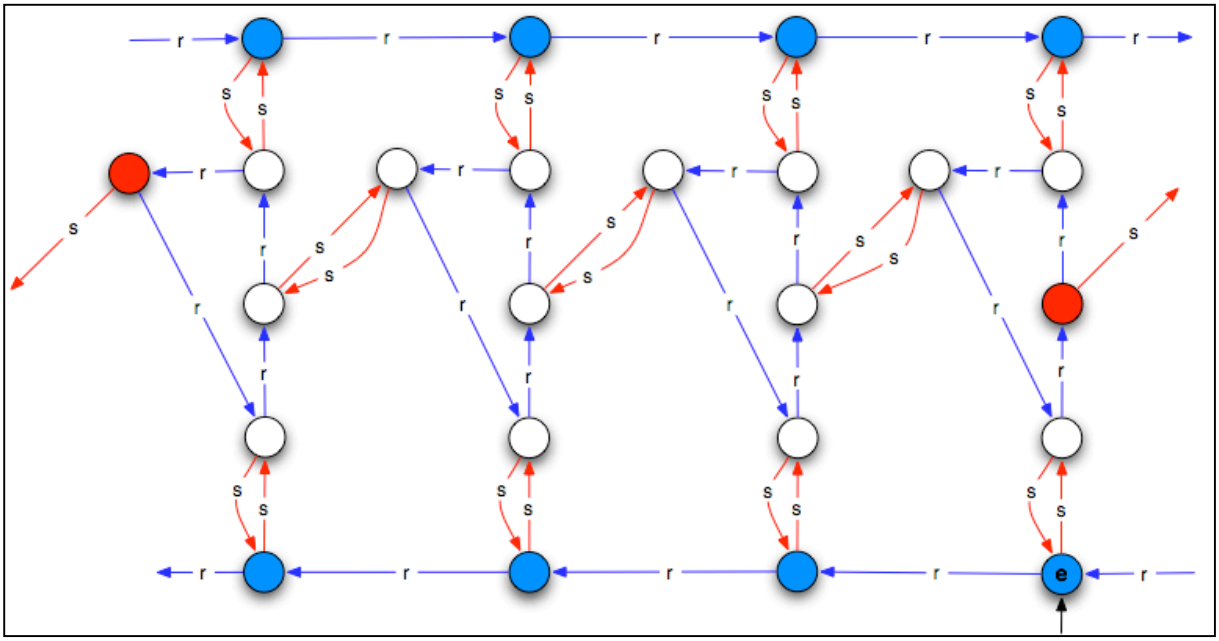


Fig. 2.13 DFA isomorphic to the symmetries of the cube

3. LOW-DIMENSIONAL NON-LINEAR DYNAMICAL SYSTEMS

This chapter reports on a part of the work performed in the area of mathematical modelling of physical systems. A great deal more was begun in this general research area than is reported here, but in the interest of reducing clutter we will discuss here only those aspects of physical systems and the underlying mathematics that seem most relevant to the problem of the self-organisation, adaptation, and self-healing of software, which as discussed in the previous chapter we can loosely refer to as "biocomputing". First, however, we need to provide some background and motivation for this work.

Background, motivation, and scope

The heart of the (natural science) work in WP18 is motivated by the belief that order construction processes in Nature are fundamentally non-linear, and that therefore we need to develop a mathematical framework that is able to give us useful insights into such a class of problems—that usually cannot be solved explicitly. This sounds challenging enough, but it is in fact just the starting point, since biological systems are further characterised by a nested hierarchy of structure and behaviour over a large number of orders of magnitude.

This deliverable does not address statistical physics, whether in its equilibrium or in its non-equilibrium forms. The latter is referred to as critical phase transitions or critical phenomena in general, and we still consider it the main mechanism for the exchange of energy and information between different scales of a system. As such, it is clearly quite relevant to the structural and functional architecture of complex systems and certainly warrants further study. However, as discussed in D18.1, in physical systems it depends on a balance between two different and opposing universal processes: the minimisation of a system's overall potential energy of interaction, which tends toward order and "crystallisation" of systems of multiple components and which is associated with low temperature; and the maximisation of the system's entropy, which simply quantifies the spontaneous tendency toward configurations of maximum probability and is associated with high temperature. In systems where a potential energy of interaction can be defined (in other words where an attraction or repulsion force exists between their components), self-organisation can be understood as the result of these two universal and spontaneous tendencies in Nature working together toward the minimisation of a more general function that combines them called the "free energy". Many, if not most, biological processes at the molecular scale (which is of principal interest in WP18) can be understood in terms of the propagation of critical phase transition boundaries in heterogenous solutions, driven by reaction-diffusion processes that minimise the free energy locally as well as globally.

However, digital information systems are abstract and no such force of interaction can implicitly be defined between their components, at any scale.

After reaching a satisfactory conceptual understanding of self-organisation in physical systems, therefore, it seemed appropriate to address the question of self-organisation at a deeper level, that could bridge the gap between the physical and the digital worlds. This required the selection of a class of physical problems that (1) seemed simpler than critical phase transitions and (2) might be easier to map to software. The problem was reduced to one of interaction between a smaller number of components, namely two, arriving at the coupled degrees of freedom (DoFs) of low-dimensional non-linear dynamical systems. The non-linear character is a necessary consequence of the presence of coupling between the degrees of freedom. This choice of problem also satisfies the second requirement, since "coupling" means exchange of energy between the DoFs, which can be treated as analogous to the exchange of information between software components ("components" can mean "agents", "programs", "applications", "services", "modules", etc). In other words, the interaction between individual components coupled at any one scale was prioritised over the interaction between sets of components coupled across different scales.

The strength of interaction between physical degrees of freedom in a mechanical system can be inferred from the rate of energy exchange or "flow" between the DoFs. Energy flux (Joules/s) is easily quantified and averaged, and it plays a similar role in mechanical systems that temperature plays in statistical systems. In fact, high (mechanical) energy leads to deterministic chaos in the former and loss of crystalline structure in the latter. Information flux (bits/s or bandwidth), on the other hand, cannot be seen as the "effect" of any recognisable "cause" playing a role remotely similar to the stiffness of a spring. Information flux is quite arbitrary and is limited only by the bandwidth capacity of the hardware. In front of such a marked difference between the worlds of physics/biology and the world of software, we were rescued by an abstract concept that is common to the mathematical models of both worlds, the concept of state. As discussed in D18.1, in fact, the exchange of energy between DoFs can be seen as coincident with the exchange of information between the same DoFs, leading to corresponding changes of state of the system (or sub-systems, depending on the focus of the analysis). Thus, the concept of state enables us to arrive at a common definition of non-linearity for physical and software systems: non-linearity involves the roughly symmetrical exchange of energy and information between components or DoFs of a system, such that the future state of any one component is influenced by the component(s) it is coupled to and, through the resulting feedback loops, by its own state.

Whereas in some cases the mathematical models of such systems, expressed as sets of coupled non-linear differential equations, are amenable to solution by global (Lie) symmetries, such regularity properties can only address part of the problem of the construction of order. Complex systems that construct order are in fact open both in terms of energy and information exchange with their environment. Because such open systems develop through various types of interaction with their environment, we could say that a good part of the information that determines their morphology is resident in their environment, and this is the main reason internal symmetries cannot be the whole story of where ordered structures come from. However, symmetries remain fundamentally important as they provide a "handle" for which we do have a reasonably well-developed mathematical theory and, as we discussed in Chapter 2, are considered important to our vision of biocomputing.

A simple example of a system built through the balance between internal symmetry and environmental conditions is the snow flake. No two snow flakes are alike since the ice accretion rate is affected by infinitesimal temperature and humidity fluctuations whose *history* is different for each snow flake as it falls toward the ground in a turbulent cloud; however, its small size relative to the length scale of temperature variation causes each of the 6 arms to experience the same conditions and therefore grow in roughly the same way. This preserves the initial 6-fold symmetry given by minimisation of (potential) energy between the first 7 water molecules that come together (as can be easily demonstrated with 7 identical coins laid flat on a table).

There is one additional concept that we need to account for the required level of complexity and that at the same time makes the system model closer to software: memory. Rocha (1998) and co-workers at Santa Fe and Los Alamos talk about "dynamical coherence" to describe the time evolution of a system as dependent on instantaneous interaction with its environment, without memory. The same concept applies also to how coupled degrees of freedom are related to each other in non-linear dynamical systems: two degrees of freedom are dynamically coherent when they are tightly coupled; when, in a sense, each is reacting to the other without the ability to set an independent agenda. Agenda is linked to agency, which leads to the complementary term "dynamical incoherence". In order to understand the process that sustains the evolution of a genome we need to take its environment into account ("evolutionary closure"). But in order to talk about the genome and its environment we must be able to distinguish between the two. The "boundary" of a system that allows it to be distinguished from its environment depends on the concept of autonomy or agency. Agency depends on the ability of the system to make decisions, i.e. to step outside of the dynamics of interaction with its environment. In physics such a system (an agent) is described as "dynamically incoherent". In computer science Von Neumann called such systems "memory-based systems", i.e. "Systems capable of engaging with their environment beyond concurrent state-determined interaction by using a stable random-access memory to store descriptions and representations of their environment" (Rocha, 2000). The former is analogous to stateless communications in networking, while the latter is analogous to stateful communications.

If we identify DNA with such a memory that enables asynchronous interaction, we can summarise the discussion so far with a set of system properties that appear to be common to the cell, a plausible software component architecture, and their underlying mathematical model:

1- stateless as well as stateful coupling between components
2- internal memory to support asynchronous interaction
3- nested hierarchical structure
4- regularity of structure and behaviour in all of the above modelled through symmetries

This chapter and the next (Chapter 4) focus on points 1 and 4 from the point of view of the physics and mathematics of continuous low-dimensional non-linear dynamical systems. Chapter 5 begins to address points 1, 2, and 4 from the point of view of computer science. Point 3 is not addressed by this deliverable beyond a few references to elementary network topology.

Interaction and coupling are regarded as embodying that aspect of open systems that can be modelled mathematically, even if in a necessarily reductionist manner. A research agenda has therefore emerged whereby the regularities in structure and behaviour observable in biology and physics are to be expressed in a mathematical formalism that can be translated into the formalism of state machines or state transition systems. Given, furthermore, that computer science has been able to develop a powerful theory to represent interactions between concurrent processes, it seems plausible that suitable constraints can be derived from the former to be imposed on the latter to enable them to exhibit dynamic behaviour analogous to biological systems.

This picture is rather optimistic, however, since while the presence of global symmetries in dynamical systems is a symptom of their integrability, their chaotic behaviour is a symptom of their non-integrability. Thus we seek non-linear models due to their flexibility and expressiveness, but not *too* non-linear lest they become completely unusable. It has been suspected for a long time, in fact, that the most interesting dynamical behaviour is carefully balanced between order and chaos.[22] The research effort required to address this class of problems mathematically is very large indeed, even at the level of an in-depth literature review, and well beyond the resources of WP18. Therefore, it seems appropriate to turn to a more intuitive approach rooted in computer science, using a more qualitative perspective on dynamical systems theory as a source of metaphors, insights, and inspiration. The motivation for forging ahead even in the absence of a mathematical theory that can provide solutions to this class of problems comes from the observation that biological systems have clearly found such solutions. Thus, where a rigorously formal approach may be insufficient, an intuitive engineering approach may succeed.

It is with this view in mind that this chapter and the next begin a discussion of non-linear dynamical systems and of the symmetry methods that can be used to find their solutions. Subsequent chapters and the on-going work in the OPAALS project will then attempt to map the intuition that we are starting to develop here to computer science.

Generalised potential of 2 DoF non-linear Hamiltonian dynamical system

Some effort was spent on analysing a long-standing class of problems in non-linear dynamics about which much has been published: a 2 DoF Hamiltonian mechanical oscillator where the coupling is carried by centrifugal and Coriolis forces. The motivation was to gain a better insight into the stability properties of this kind of oscillator. Three closely related such problems were examined: Chaotic Turntable, Spring (or Elastic) Pendulum, and Centrifugal Oscillator. This work is quite elementary and is not meant in any way to represent the state of the art in non-linear dynamics. It is only meant as an aid to the conceptualisation of non-linear behaviour. More work will need to be expended (in OPAALS) if the progress made on the modelling of discrete and formal systems justifies it.

[22] See for example the extensive literature since the 1940s in electronic engineering, astrophysics, music, and non-linear dynamics on (1/*f*)-noise.

Chaotic Turntable

Fig. 3.1 shows a chaotic oscillator that is discussed in (Cusumano and Moon, 1995) and that was developed to model the behaviour of a thin elastic beam clamped at one end, subject to bending and torsional displacements, and allowed to vibrate freely at the other end. We are interested in the free as opposed to the forced response of this system. The system is a slotted turntable pivoted around its central axis (pivot not shown) that can be considered to be a simplified model of a 2-D "slice" of the elastic beam. Thus, the first spring (k_1) and the moment of inertia J relate to the torsional motion of the beam, while the second spring (even though the picture shows two springs in the slot this is done as an aid to physical intuition but is equivalent to a single spring) and the mass m relate to the bending motion. The variables θ and s measure angular and linear displacement of the turntable and of the mass from the equilibrium positions of the two springs, so they can be regarded as two separate oscillators that are coupled through centrifugal and Coriolis forces. There is no friction and the turntable is assumed to be horizontal, so gravity does not play a role.

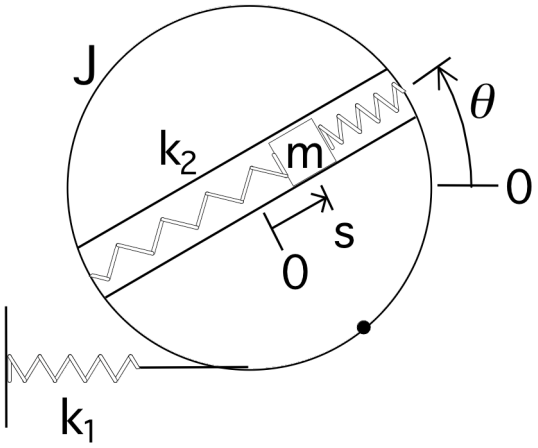


Fig. 3.1 A chaotic oscillator

It helps to think about this system intuitively before doing any maths. If the turntable and the mass are displaced from their equilibrium position and released (from rest, for simplicity), they will start oscillating back and forth and they will keep going forever since there is no friction. The most important trend that we can recognize immediately is that if the initial displacements are small then the motion of each oscillator (mass and turntable) will be quite close to pure sinusoids, and the two DoFs will behave essentially as if they were two separate and independent simple harmonic oscillators (SHOs). As the initial displacements are increased, however, the motion will become increasingly complicated until it will be completely chaotic and random. Since there is no friction, we know that the total mechanical energy (kinetic + potential) is conserved, and therefore the initial displacements store potential energy in the springs that represents the total energy budget of the system and stays at the same value ever after. So the important conclusion is that as the energy of a system increases, the likelihood that its behaviour will turn chaotic increases, which is quite universally true.

A hint at how this might happen is provided by the following facts. We said that at very low energy the two degrees of freedom are not "communicating." One can actually show that each oscillator, as long as the amplitude of its oscillations is small, will conserve its own energy, and will not share it with the other oscillator. As the amplitude increases, on the other hand, two physical mechanisms become important that cause the energy in one oscillator to be "sucked up" by the other, only to be given back shortly thereafter, and so on. The two mechanisms go by the names of centrifugal and Coriolis forces, and are both a direct consequence of the unwillingness of inertia to change its current state of motion, whatever that state of motion may be (Newton's First Law). Centrifugal force is quite familiar to most people and can be easily seen to cause the mass to absorb energy as the turntable spins around rapidly, throwing the mass away from the centre. Since as the mass moves radially outward it moves from regions of lower

tangential velocity to regions of higher tangential velocity, the side of the slot will push on it to speed it up; this is the Coriolis force, which can be thought of as a "leak" of kinetic energy from the turntable "into" the mass's. If the mass is, instead, moving toward the centre, it will be moving from a region of higher tangential velocity to a region of lower tangential velocity, thereby pushing on the slot. This is still the Coriolis force, but in this case the mass is giving some of its energy to the turntable. In a chaotic system, therefore, the energy is sloshing about between the various degrees of freedom, much as water in a bathtub mounted on a see-saw sloshes about as the see-saw goes up and down—thereby affecting the motion of the see-saw itself, which in turn affects the motion of the water, etc.

Interesting things happen when a system is pushed just beyond the linear regime, but not so far as to go completely wild. Another view of this is provided by information theory which says that white noise (i.e. chaotic motion) holds an enormous amount of information, while periodic signals are very information-poor. If there is too much information we approach a statistical, random, stochastic process that maximizes entropy and that we can't keep up with, while if we go the other way symmetry and order reign supreme and things get very predictable. Biological life-forms are systems balanced between order and chaos, where new information is always readily available but sufficient order and symmetry remain to get something done. The energy that pushes life-forms toward chaos comes from sunlight and cascades into its many different forms down to random molecular motion making all the machinery around us breathe and grow in the process.

The (non-integrable) equations of motion of the chaotic turntable are:

$$\begin{aligned} \frac{d}{dt}\left[\left(1+s^2\right)\dot{\theta}\right] + \theta &= 0 \\ \frac{d}{dt}\left[\dot{s}\right] + \left(P-\dot{\theta}^2\right)s &= 0, \qquad P=\frac{J/mR^2}{k_1/k_2} \end{aligned} \tag{3.1}$$

Spring Pendulum

The next problem, shown in Fig. 3.2, is qualitatively very similar and has received more attention than the former (Olsson, 1976; Falk, 1978; Cuerno, Ranada, and Ruiz-Lorenzo, 1992; Davidovic, Anicin, and Babovic, 1996; Georgiou, 1999).

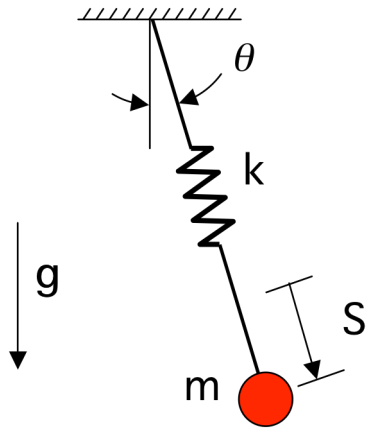


Fig 3.2 Spring Pendulum (or Elastic Pendulum)

This system can easily be built in the lab and demonstrated to behave as an autoparametric oscillator (Tondl, Ruijgrok, Verhulst, and Nabergoj, 2000) when the ratio of the spring natural frequency is twice the pendulum natural frequency.

The equations of motion of the spring pendulum are:

$$\begin{aligned} \frac{d}{dt}\left[(1+s)^2\dot{\theta}\right] &+ (1+s)\sin\theta &&= 0 \\ \frac{d}{dt}\left[\dot{s}\right] &+ \left(P-\dot{\theta}^2\right)s &&= \cos\theta + \dot{\theta}^2, \qquad P=\frac{k/m}{g/l} \end{aligned} \tag{3.2}$$

Centrifugal Oscillator
The centrifugal oscillator was derived as a simplification to the spring pendulum. In fact, given that gravity is missing the troublesome trigonometric terms do not appear. As shown in Fig. 3.3, the return force analogous to gravity for the pendulum is provided by a linear spring acting purely tangentially. The guide and the springs are massless.

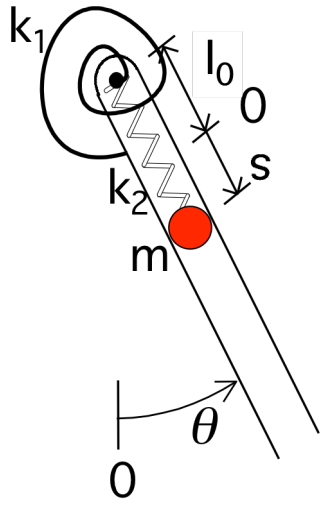


Fig 3.3 Centrifugal Oscillator

The equations of motion are:

$$\begin{aligned} \frac{d}{dt}\left[(1+s)^2\dot{\theta}\right] &+ K\theta &&= 0, \qquad K=k_1/k_2 \\ \frac{d}{dt}\left[\dot{s}\right] &+ \left(1-\dot{\theta}^2\right)s &&= \dot{\theta}^2 \end{aligned} \tag{3.3}$$

The centrifugal oscillator is simpler than the spring pendulum. In an attempt at visualising its total qualitative behaviour, a *Mathematica* animation of its time-dependent effective (or generalised) potential was developed. The effective potential was originally defined in orbital mechanics and represents the negative integral of the force field acting on a particle. This is a simple reversal of the fact that in a conservative system the force is the negative gradient of the potential. In the equations above the force can easily be recognised by isolating the time-rate-of-change of the generalised momenta:

$$\begin{aligned} F_\theta &= -K\theta \\ F_s &= \dot{\theta}^2 - \left(1-\dot{\theta}^2\right)s \end{aligned} \tag{3.4}$$

Each of these expressions can then be equated to the negative gradient of a potential surface in the two independent directions:

$$\begin{aligned} -\frac{\partial U}{\partial \theta} &= -K\theta \\ -\frac{\partial U}{\partial s} &= \dot{\theta}^2 - \left(1-\dot{\theta}^2\right)s \end{aligned} \tag{3.5}$$

Integrating,

$$U(\theta,s) \;=\; \frac{1}{2}K\theta^2 \;+\; \frac{1}{2}s^2 \;-\; s\left(1+\frac{1}{2}s\right)\dot{\theta}^2 \tag{3.6}$$

Since both θ and s are functions of time, whose solution is given by the non-integrable equations of motions (3.3), the effective potential can be visualised over a certain time-window only after a numerical solution has been obtained for the system over the same time-window. Fig. 3.4 shows a sketch of the effective potential. This figure highlights how the effective potential provides an intuitive way to visualise the motion of the mass as a ball rolling in a parabolic trough under the action of gravity. The trough is not fixed, however, as can be inferred from Eq. (3.6). Although the θ-coordinate has been "unwrapped" to become linear, the top view of this motion is topologically equal to the actual motion of the mass in the physical plane.

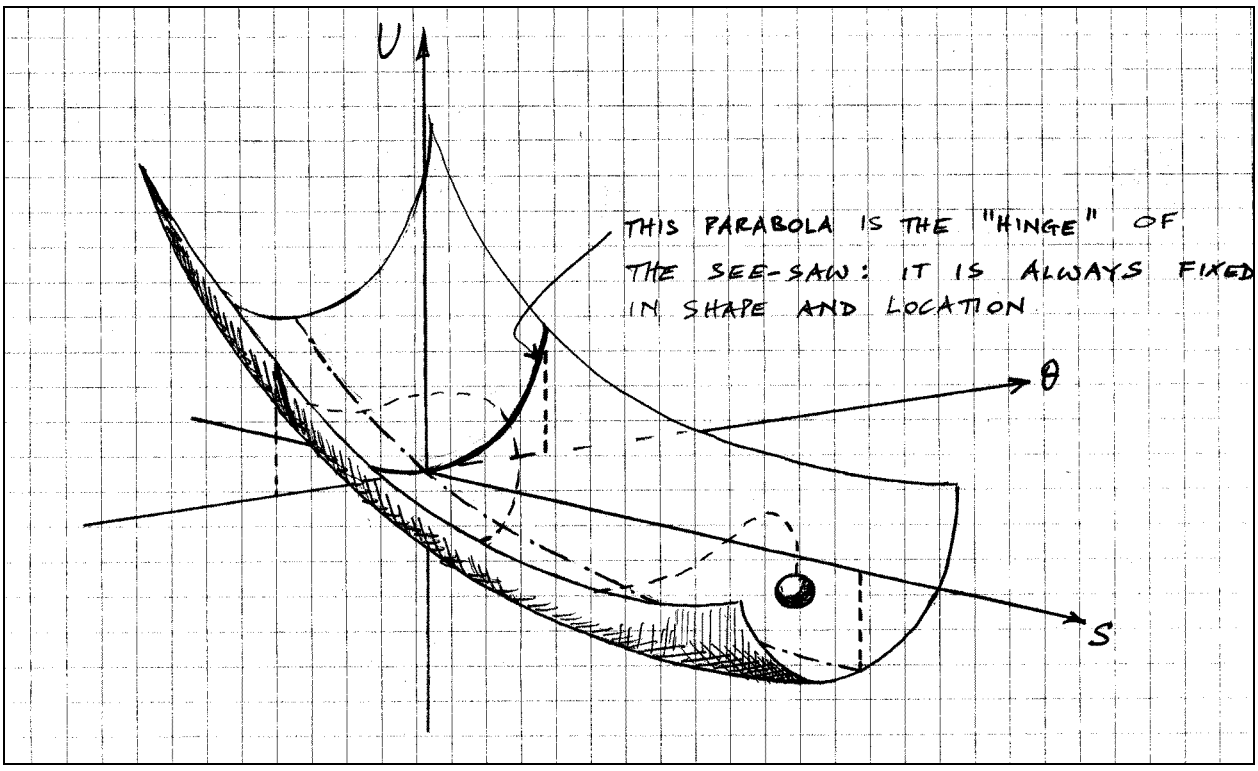


Fig. 3.4 Sketch of the effective potential of the centrifugal oscillator

The stability of time-dependent interactions

The radial equation for the turntable oscillator can be understood as a simple harmonic oscillator whose spring stiffness varies as a function of time. For low system energy the variation of θ w.r.t time is

approximately sinusoidal, meaning that Eq. (3.1b) is similar to a Mathieu equation (linear second-order ODE with periodic coefficients). Two non-contiguous snapshots of a *Mathematica* animation of the time-dependent generalised potential of the chaotic turntable are shown in Fig. 3.5. In these figures the mass is shown as a red dot "rolling" on the smooth potential surface, without friction and under the action of gravity. The past trajectory of the ball is shown as a black line drawn on the surface,[23] highlighting the rather regular behaviour of the mass in spite of the non-linear character of the equations and of the markedly time-dependent geometry of the potential. For higher system energies the motion is not stable any longer and the trajectory turns quite chaotic, as we expect.

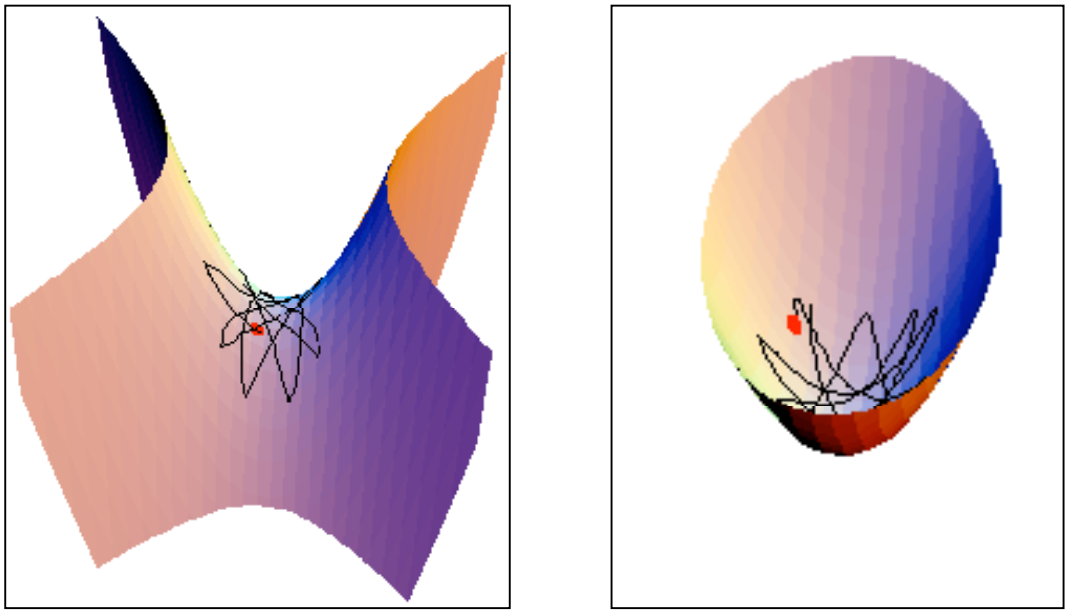


Fig. 3.5 Two snapshots of the effective potential animation of the chaotic turntable

The hyperbolic potential surface of Fig. 3.5a is clearly unstable, in the sense that if it were static the ball would roll away from the centre, without coming back. The stabilising effect of the time-dependence of the potential is reminiscent of a Paul trap, used in experimental physics to trap ions at the centre of an oscillating quadrupole potential that can in fact be visualised as a oscillating hyperboloid.

The interaction between the two degrees of freedom of this system helps us conceptualise how the behaviour of interdependent biological sub-systems leads to stable metabolic behaviour as long as the dynamic character is maintained. A similar concept of dynamic stability may be necessary to reach quickly responsive and adaptive yet useful behaviour out of a set of concurrent software processes.

These non-linear oscillators motivated the search for a solution method that could uncover the source of the regularities observed. Group-theoretic methods, and in particular Lie groups, are the only methods we have for finding the exact solution to non-linear problems, when it exists. The systems discussed in this chapter, however, are not integrable, which means that the regularities observed may be analysed through approximate or partial solution techniques, as discussed in some of the references given. None-the-less, a deeper understanding of symmetries seems essential to progress even at only an intuitive level toward understanding self-organisation. The next chapter documents the work that has been started in this direction.

[23] The *Mathematica* package CurveOnSurface was developed by Gianluca Gorni at the University of Udine www.dimi.uniud.it/~gorni/

4. Basic Facts about Symmetries of ODEs

This chapter documents work begun at the beginning of the project that aimed to develop an analytical tool for systems of ordinary differential equations (ODEs) in the form of a *Mathematica* notebook. The objective of the tool was to search for the symmetries of given equations or sets of equations through an automated symbolic manipulation algorithm. Such algorithms are well known in principle but not necessaritly trivial to implement. The effort stopped well before this ambitious goal due to lack of resources and different project priorities, namely a greater emphasis on discrete systems. The fact that discrete symmetries and Galois fields still appear to offer the only connecting bridge between computer science and gene expression and morphogenetic processes in Nature, however, justifies the reporting of this effort in this deliverable, even if the work reported is concerned with continuous symmetries. Group theory is so abstract, in fact, that exposure to carefully explained elementary concepts is believed to be helpful for any non-mathematician approaching this subject. Clearly much work remains to be done, but if this deliverable is able to argue convincingly about areas and methodologies of research that have not thus far received as much attention as they deserve then it will have achieved something useful---even if not immediately implementable.

This chapter is largely based on Peter Hydon's excellent introductory book on symmetries of ODEs (Hydon, 2000). Not everything discussed is from his book, but a good part. In general concepts and derivations are expanded and explained to an even more elementary level than Hydon's. The theory of Lie groups is considered to be well understood and well developed. No attempts are made at referencing all the work that has gone before, given the elementary nature of the material discussed.

■ Objective

Symmetries are useful for solving non-linear ODEs. Their utilisation is not trivial in the sense that many additional facts about mappings, functions, vector calculus, and higher-dimensional geometries need to be understood and relied upon. This makes the theory of Lie symmetries very rich. These notes summarise the basic facts and illustrate them by means of examples, taken mainly from Hydon. The basic facts, however, will eventually be reformulated in a more general notation in order to be able to analyse systems of two coupled ODEs, and in particular the centrifugal oscillator presented in the previous chapter. The analysis of the centrifugal oscillator is motivated by the desire to understand meta-stability (as defined in Chapter 3), to define it more precisely, and possibly to develop a quantitative criterion for detecting it. If meta-stability is related to symmetries, we may need to develop an approximation of the global symmetries discussed here in order to model systems that exhibit regular, meta-stable behaviour over only a finite range of system energy such as, for instance, biological systems. The ultimate objective, therefore, is to develop the simplest possible model of a dynamical system that exhibits a meta-stable behaviour similar to the meta-stable behaviour characteristic of biological systems.

■ Table of Contents (section numbering in this chapter is self-contained)

4. Symmetries of differential equations and groups
5. The symmetry condition
6. Orbits
7. Canonical coordinates
8. Linearised symmetry condition
9. A detailed example
10. Symmetries and standard methods
11. Integrating factor

(The following sections were originally planned but are not covered by this chapter)

II. Higher-order ODEs
1. Infinitesimal generator
2. Symmetry condition for higher-order ODEs
3. Linearised symmentry condition for 2nd-order ODEs
4. Reduction of order

III. Variational symmetries

IV. Systems of ODEs

V. The Centrifugal Oscillator

■ 1. Concept

We are all familiar with reflection symmetry in the everyday sense, as used in art, architecture, design, etc. In mathematics we generalise the concept by saying

Symmetry is invariance with respect to some transformation (1.1.1)

In this way symmetry is used as the property of an object. In the conventional sense an object is symmetrical if it is invariant under reflection about its vertical central axis. Objects can be symmetrical with respect to more general transformations, e.g. snowflakes with respect to 60 deg rotations. For our purposes, however, this terminology is confusing. Rather than identifying symmetry with a ***property*** of an object under some transformation, we will identify symmetry with the ***transformation itself***. We thus come to the alternative definition:

A symmetry is a transformation that leaves an object invariant (1.1.2)

We can then talk about objects that "admit" symmetries or "possess" symmetries if transformations, maps etc can be found that leave them invariant. These two meanings will keep playing tricks with our minds because, no matter how carefully we define the second, the everyday meaning and interpretation of the word will sneak into our arguments and our thinking when we least expect it. As long as we are aware of this and can recognise it when it happens, however, it should not cause any problems.

These notesfocus on a particular type of symmetry called continuous (as opposed to discrete) or Lie symmetry, from Sophus Lie (pronounced "Lee"), a Norwegian mathematician who lived in latter part of Nineteenth Century.

■ 2. Concept in more mathematical terminology

There is another area of our mind that will play tricks on those of us who are not trained and active mathematicians: the area that conceptualises spaces of different dimensions. The study of symmetries exacerbates this problem because it is fundamentally based on the mixing, within the same conceptual and algebraic constructs, of functions defined in spaces of different dimensions. This means that the mixing of spaces of different dimensions, even if low, presents a more subtle and devious set of potential trip-ups for the deductive machinery of our mind than the straightforward difficulty of visualising a geometrical object in a space of 4 dimensions. The result is a

perception that this branch of mathematics is abstract and difficult, which is not entirely the case. Let us therefore develop the terminology and the concepts gradually.

Take a function

$$y = f(x) \qquad \text{or} \qquad x \xrightarrow{f} y \qquad (1.2.1)$$

Its shape or graph is (ignore the arrows for now)

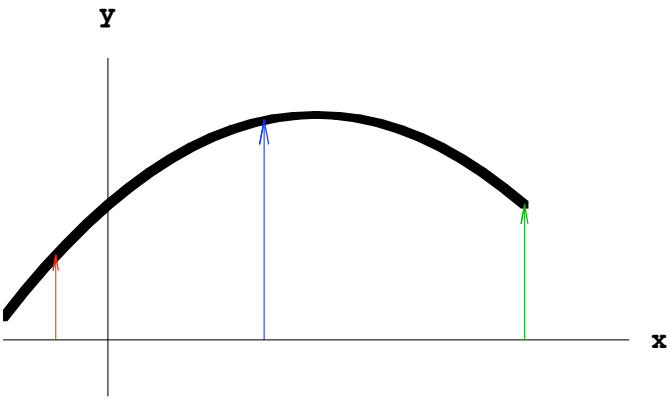


Fig 1. 2D graph of 1D function

Having defined one-dimensional functions, let's generalise to 2D maps:

$$(x, y) \xrightarrow{\text{map}} (u, v) \qquad (1.2.2)$$

Its shape cannot be drawn because it requires 4 dimensions; all we can draw is:

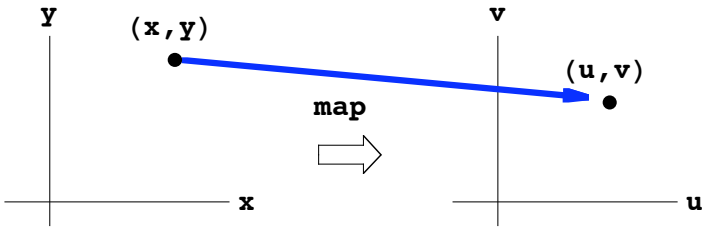


Fig 2. 2D map

Every point (x, y) is mapped to a point (u, v). No shape can be drawn. Now we notice that having defined a 2D map we have allowed the possibility of introducing additional structure: in other words, over and above the map we can also define a relationship between x and y, which will be mapped to a relationship between u and v: This relationship is indicated by a "V" pointing at the two variables that it connects. This is not a formal symbol, it is just a simple graphical aid.

$$\underset{\vee}{(x,\ y)} \xrightarrow{\text{map}} \underset{\vee}{(u,\ v)} \qquad (1.2.3)$$

the corresponding picture is

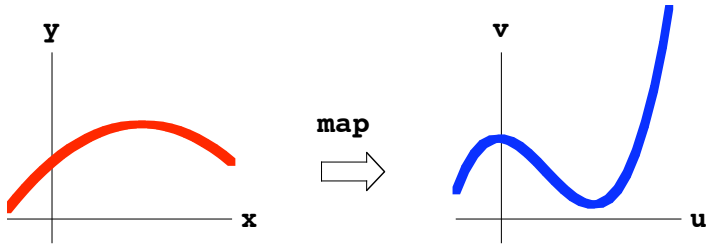


Fig 3. 2D map of 1D function

We can now say that

a symmetry is a map that leaves the shape of a function invariant (1.2.4)

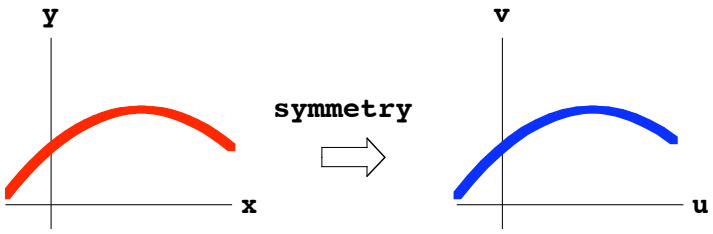


Fig 4. Symmetry of 1D function

From this we observe that to find a symmetry for a (1-D) function f(x) we need to pose the problem in a space of higher dimension (2-D).

■ 3. Visualisation of maps and their graphs in different dimensions

Before we move to differential equations, some additional preliminary concepts are needed. Specifically, it will be helpful to have a clear picture in mind of the different ways maps can be represented graphically in different dimensions. Take the graph in Fig 1, for example. Even though it is not easier to see and understand, the same information can also be drawn using only one dimension, as shown in Fig. 5.

Fig 5. 1D function graphed in 1D space

The length of the arrows in this figure represents the magnitude of the function, i.e. the value y. Each arrow is drawn with its base at the point x that is used as an input to the function y(x) to calculate the value of y. A conceptually more useful 1-D depiction is to draw an arrow from value of x to the value of y, where now y is referenced to the origin rather than to x, as we had implicitly assumed for Fig 5. Fig 6 shows the corresponding graph:

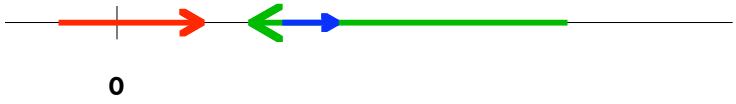


Fig 6. A better way to graph a 1D function in 1D space

It is not necessarily the case that this was better because it is easier to see. The arrows are drawn in different colours to make it easier to interpret the graph, but in any case either method can lead to very confusing graphs depending on what function is being graphed. The reason the second way is better, in addition to being more consistent with the common interpretation of y(x) as a map $\mathbb{R} \longrightarrow \mathbb{R}$, is that it implicitly introduces the concept of vector. If x and y are considered two separate 1-D vectors, drawn from the origin to the values of x and y, respectively, then each of the arrows drawn represents the vector *difference* [y - x], which is an arrow drawn from the "tip" of x to the "tip" of y. This concept will be quite useful in 2-D.

For the sake of completeness, the notion of a map from every point in one space to a point in another space, shown in Fig 2 for 2-D, can be rendered also in 1-D:

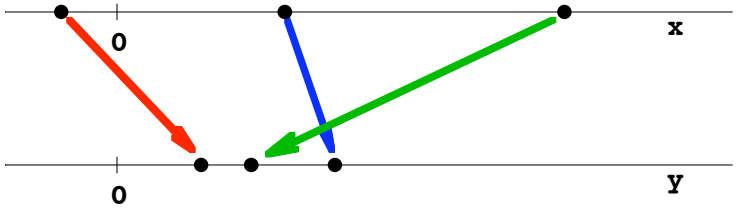


Fig 7. 1D map

Now we should explore the "additional structure" shown in Fig 3 a little more quantitatively. This is mainly so we can arrive at a visualisation of 2-D maps that is intimately related to the symmetry discussion below. So, let us take for instance the following linear, discrete map:

$$u(x, y) = 3x + 2y \qquad (1.3.1)$$
$$v(x, y) = 5x - 6y \qquad (1.3.2)$$

This is called a discrete map because the new point (u, v) has a value that could be quite different to the starting point (x, y). Each point in the xy-plane, therefore, "jumps" to a different point in the uv-plane. One way we have of visualising this map is to look at its effect on a curve such as $y(x) = x^2$. Plugging this into Eqs (1.3.1) and (1.3.2) and eliminating x we get two branches of the same function v(u):

$$v(u) = -\tfrac{15}{4} \pm \tfrac{5}{4}\sqrt{9 + 8u} - 6\left(\tfrac{9}{16} \mp \tfrac{3}{8}\sqrt{9 + 8u} + \tfrac{1}{16}(9 + 8u)\right) \qquad (1.3.3)$$

So the two functions can be graphed as follows:

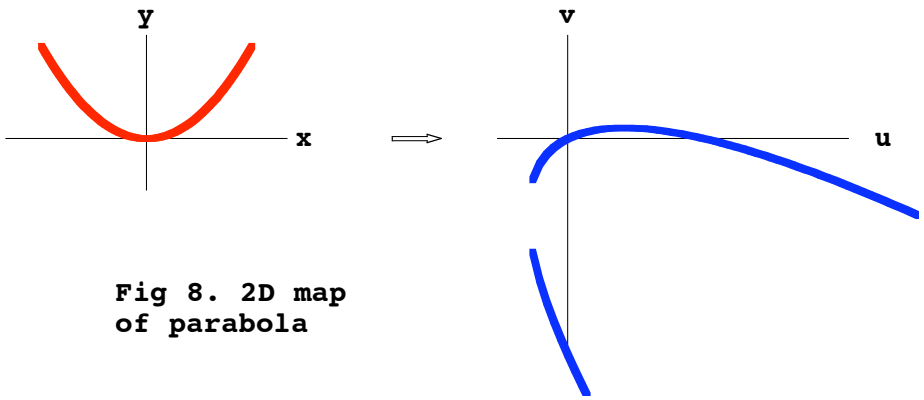


Fig 8. 2D map of parabola

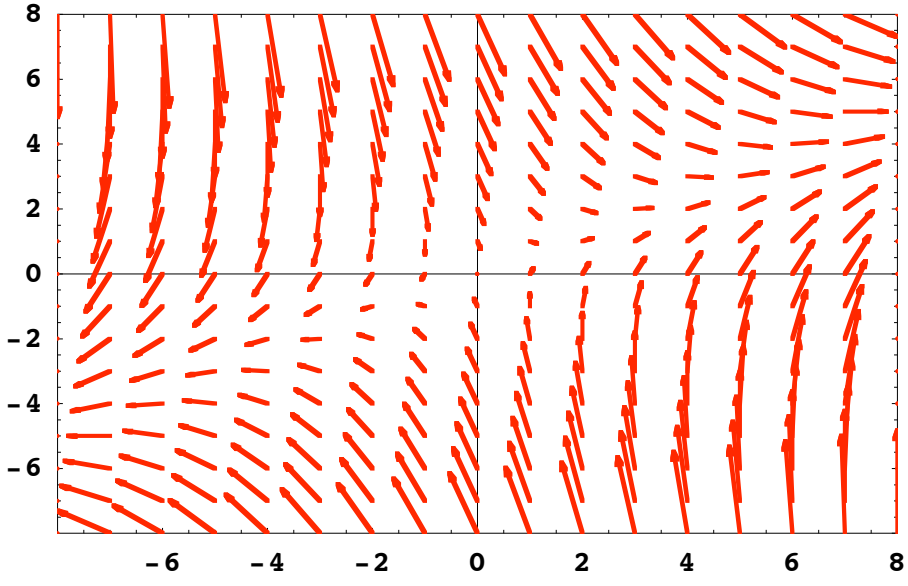
This looks like a rigid rotation of the original curve. In order to visualise the 2D map without the benefit of a 4D space we can use the trick shown in Fig 6 and depict the same 2-D map given in Eqs (1.3.1)-(1.3.2) as a vector field in the x-y plane. To do this we first note that both the (x, y) and the (u, v) coordinates are referenced to their origin. We then imagine that the points (u, v) are drawn in the x-y plane, thereby performing what is called a mapping from the plane to the plane. At this point, as we did in Fig 6, rather than plotting each point (x, y) or (u, v) as a vector from the origin to (x, y) or (u, v), we take their ***vector difference*** [(u, v) - (x, y)]. This gives a vector from the tip of (x, y) to the tip of (u, v) that we can draw explicitly as an arrow. This representation is more useful because it shows us how the point (x, y) has moved to (u, v), which corresponds to how we think of a map intuitively. For any input point (x, y), therefore, what we are actually plotting in Fig. 9 are vectors whose x and y components are given by:

$$x\_component(x, y) = [3x + 2y] - [x] = 2x + 2y \quad (1.3.4)$$

$$y\_component(x, y) = [5x - 6y] - [y] = 5x - 7y \quad (1.3.5)$$

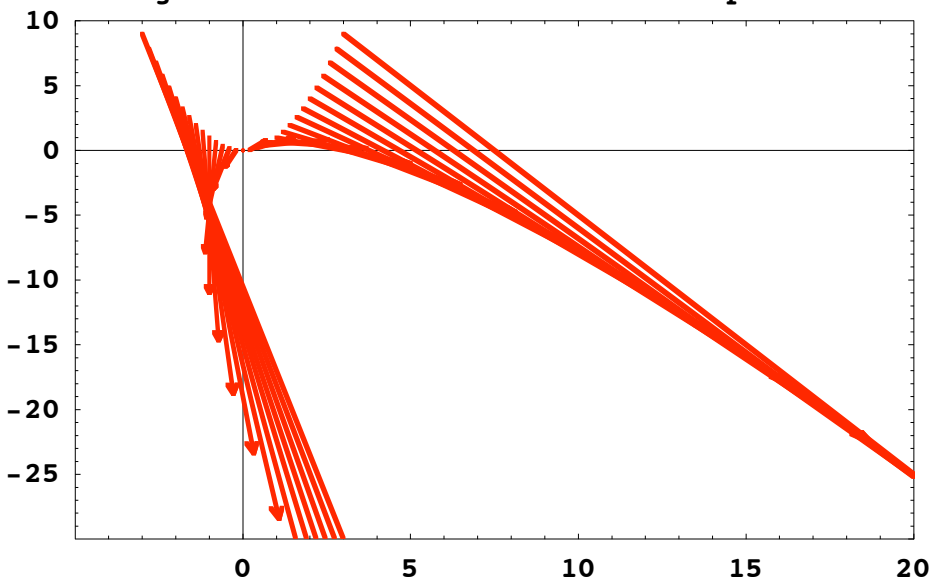
Surprisingly the resulting vector field does not look like a rotation at all.

Fig 9. 2D map as a 2D vector field



Just to make sure, let's verify that this vector field is consistent with the map of the parabola shown in Fig. 8. Fig. 10 shows that it does in fact work.

Fig 10. Action of vector field on parabola



Symmetries as defined so far can thus be visualised as vector fields that leave the shape of a particular function invariant. In Fig 9 it's pretty clear that the map given by Eqs (1.3.1)-(1.3.2) may be a symmetry of two straight lines passing through the origin and of slope = -4 or 1/2, approximately, but certainly not of the parabola.

Another point that may be useful later is that there is an apparent similarity between the 2-D vector field depicted in Fig 9 and the gradient of a scalar function. Could it be that any 2-D map can therefore be visualised as the gradient of some other scalar function? Not quite. Since we know that the curl of a gradient is always zero (this is proven, in Section 11), we can use this fact as a condition: a vector field can only be regarded as the gradient of some scalar (potential) function if its curl is zero. In the case of the map we chose, Eqs (1.3.1)-(1.3.2), the curl of its corresponding vector field is not zero:

$$\frac{\partial}{\partial x}(2x + 2y) - \frac{\partial}{\partial y}(5x - 7y) = 9 \qquad (1.3.6)$$

Later we will examine some symmetries of ODEs with these concepts in mind.

■ 4. Symmetries of differential equations and groups

This section begins to borrow liberally from Hydon.

The usefulness of the definition of symmetry given in Section 2 for solving ODEs is not readily apparent. Although as we shall see later it is in fact relevant also for differential equations, there is a slight variation on the definition whose relevance is more obvious:

The symmetry of an ODE is a map that leaves the form of the ODE invariant (1.4.0)

To see why this definition is not as different to (1.2.4) as it may appear will require some work. The end result will be to reach a conceptual framework for thinking about differential equations that makes them appear not so different to functions as we normally think of them, making it easier to visualise the ODEs themselves.

The property of an ODE of retaining its functional form under the action of a particular transformation is called **conformal invariance**. As a corollary of (1.4.0), a symmetry maps solutions into solutions, which is not necessarily easy to see. Rather than prove it in general, therefore, let's explain it more carefully. We tend to think of maps as

$$(x, y) \xrightarrow{\text{map}} (u, v) \tag{1.4.1}$$

or

$$u = u(x, y) \quad \text{and} \quad v = v(x, y), \tag{1.4.2}$$

which is certainly correct. However Lie added a twist: he made the map depend on one (or more) additional parameter:

$$u = u(x, y; \epsilon) \quad \text{and} \quad v = v(x, y; \epsilon), \qquad \epsilon \in \mathbb{R} \tag{1.4.3}$$

He defined the dependence on the parameter ϵ in such a way that

$$\text{when } \epsilon = 0, \qquad (u, v) = (x, y) \tag{1.4.4}$$

In these notes we will discover why leaving the form of the ODE invariant matters and that, when such a symmetry can be found for an ODE, its solution follows (or at least a significant simplification). For our present purposes we will not look at PDEs, although the method is easily extended. Let's take an illustrative example (from Hydon):

$$\frac{dy}{dx} = \frac{2y}{x} \qquad \frac{dy}{y} = 2\,\frac{dx}{x} \tag{1.4.5}$$

$$\ln y = 2\ln x + c \qquad y(x) = c_1\, x^2 \tag{1.4.6}$$

This ODE happens to have the following symmetry, which is a scaling:

$$(u, v) = (e^{\epsilon} x, e^{-\epsilon} y) \tag{1.4.7}$$

Let's not worry about how we might derive or even verify this fact for the moment. The main concern is to clarify how solutions can be mapped into solutions in any way which is different to a trivial mapping. To see this we need to apply the map (1.4.7) to map y(x) into v(u), and then show that v(u) is still a solution of the ODE (1.4.5). This is done in a few easy steps. First solve for x from (1.4.7):

$$x = e^{-\epsilon} u \tag{1.4.8}$$

then (1.4.6) becomes,

$$y = c_1 (e^{-\epsilon} u)^2 = c_1\, e^{-2\epsilon}\, u^2 \tag{1.4.9}$$

So that, plugging (1.4.9) into (1.4.7),

$$v = e^{-\epsilon} [c_1\, e^{-2\epsilon}\, u^2] \tag{1.4.10}$$

or,

$$v(u) = c_1\, e^{-3\epsilon}\, u^2 \tag{1.4.11}$$

To see how (1.4.6) is mapped to (1.4.11) it is easier to think about a mapping from the plane to the plane, as shown below.

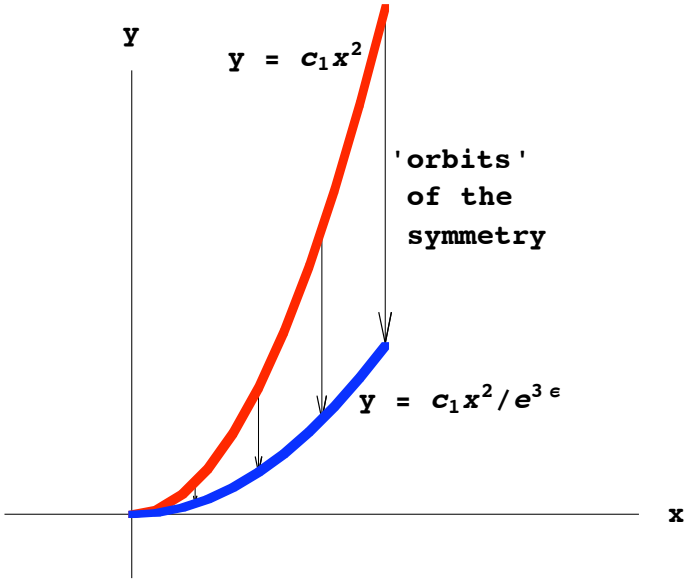


Fig 11. Action of symmetry of an ODE on its solution

Given that ϵ acts as a parameter, we can see how (1.4.11) satisfies the same ODE while being a different function: we have taken advantage of the "degree of freedom" afforded by the arbitrary constant of integration to make it incorporate also the parameter ϵ.

The theory of symmetries of ODEs is generally referred to as the Theory of Lie Groups. The group concept therefore is fundamental and can be described simply as follows: A group is a set together with an operation between its members such that

- the result of the operation is still a member of the set (set is closed w.r.t. the operation)
- the operation satisfies the associative condition
- the set has an identity element
- every element has an inverse

These rules are very general and abstract. In our case they apply as follows:

- the set is a set of one-parameter transformations (or maps), such as Eq (1.4.7)
- each member of the set is identified with a particular value of ϵ
- the operation is the composition of transformations (of course belonging to the same set)
- the identity element is given by $\epsilon = 0$
- the inverse is given by $-\epsilon$
- composing a transformation (i.e. ϵ_1) with another (i.e. ϵ_2) yields a member of the same set (i.e. $\epsilon_3 = \epsilon_1 + \epsilon_2$, in the case of an exponential map)

The above rules are common to all groups. What makes Lie groups different is the fact that ϵ is a real number. Thus, one can define "infinitesimal transformations" and can apply rules from calculus to obtain specific results, as will be shown below. For now it is sufficient to understand that the consequence of this fact is that the repeated application to the same curve of two or more successive transformations that are infinitesimally close to each other causes that curve to be mapped smoothly to a new curve. Lie groups are therefore also called "continuous groups" and the corresponding transformations are also called continuous transformations. We can in fact imagine the lower blue curve splitting smoothly from the red upper curve and moving downward as ϵ increase from 0 gradually. The path traced by a point under the smooth action of the symmetry is called an "orbit" of the symmetry. In this case the orbits of the symmetry are vertical lines pointing downward, as shown in Fig 11.

■ 5. The symmetry condition

This section begins to show how symmetries help solve ODEs. It does so by relying on the symmetry condition, which is derived by enforcing conformal invariance. In this chapter we work only with ODEs of the first order:

$$\frac{dy}{dx} = \omega(x, y) \tag{1.5.1}$$

We seek a map (u(x, y), v(x, y)) (where ϵ is implied) such that, in the new coordinates,

$$\frac{dv}{du} = \omega(u, v) \tag{1.5.2}$$

ω is the same in both equations, indicating the same functional form. Note that the term "symmetry condition" is a good example of how the two interpretations of the word Symmetry discussed in Section 1 can sneak into the mathematics through our common use of language, potentially causing confusion. What the term "symmetry condition" says literally is "condition for symmetry", tacitly implying that symmetry is meant as "invariance under some transformation", i.e. as a property of the ODE; whereas what it is ***intended*** to mean is "invariance condition that enables us to derive a symmetry", i.e. a transformation between two separate ODEs. The main result of this section will be an equation that enables us to derive a symmetry in the second sense, meaning a map that leaves the ODE invariant.

Eq (1.5.2) is deceptively simple, meaning that it is not so obvious how a differential equation might be transformed by a map of its variables. Assuming we have a transformation u = u(x, y) and v = v(x, y) and assuming it is invertible, we can solve for x(u, v) and y(u, v) and substitute them into ω(x, y) to obtain ω(u, v). However, it is not at all intuitively clear how the derivative dy/dx might be transformed by the map into dv/du. To arrive at a function that represents this transformation we start by noticing that the function ω(x, y) is clearly visualisable as a surface defined on the x-y plane. Preserving invariance of the ODE, therefore, means that the mapping needs to move points that belong to this surface to different points that *still* belong to the same surface. This geometrical requirement, however, causes an apparent difficulty since such a map necessarily needs to be defined from $\mathbb{R}^3$ to $\mathbb{R}^3$, whereas a symmetry as we have defined it so far is a map from $\mathbb{R}^2$ to $\mathbb{R}^2$. More precisely, if we wish to map the surface z = f(x, y) to another surface w = f(u, v), we need to define a transformation

$$T_3: \begin{cases} u = u(x, y, z) \\ v = v(x, y, z) \\ w = w(x, y, z) \end{cases} \tag{1.5.3}$$

whereas a symmetry as defined here for 1st-order ODEs is a transformation

$$T_2: \begin{cases} u = u(x, y) \\ v = v(x, y) \end{cases} \tag{1.5.4}$$

The trick lies in the fact that the third variable, dy/dx for z in Eq (1.5.1) and dv/du for w in Eq (1.5.2), is derivable in both equations from a knowledge of the first two, through the derivative. Since we cannot talk about dy/dx without implicitly assuming a relationship y(x) between y and x, we see that the definition of a symmetry of an ODE necessarily involves the "additional structure" of Section 2. This is not so surprising since after all an ODE implies its solution, which is exactly the same y(x).

Now dv/du can be found in a couple of similar ways. The more straightforward divides the total differentials:

$$dv = \frac{\partial v}{\partial x} dx + \frac{\partial v}{\partial y} dy \tag{1.5.5}$$

$$du = \frac{\partial u}{\partial x} dx + \frac{\partial u}{\partial y} dy \tag{1.5.6}$$

$$\frac{dv}{du} = \frac{\frac{\partial v}{\partial x} dx + \frac{\partial v}{\partial y} dy}{\frac{\partial u}{\partial x} dx + \frac{\partial u}{\partial y} dy} = \frac{\frac{\partial v}{\partial x} + \frac{\partial v}{\partial y}\frac{dy}{dx}}{\frac{\partial u}{\partial x} + \frac{\partial u}{\partial y}\frac{dy}{dx}} = \frac{v_x + v_y y'}{u_x + u_y y'} \tag{1.5.7}$$

The second way (Hydon's) is more easily extendable to higher-order ODEs. Using the total derivative with respect to x,

$$D_x = \frac{\partial}{\partial x} + \frac{\partial}{\partial y}\frac{dy}{dx} + \frac{\partial}{\partial y'}\frac{dy'}{dx} + \ldots$$
$$= \frac{\partial}{\partial x} + y'\frac{\partial}{\partial y} + y''\frac{\partial}{\partial y'} + \ldots \quad (1.5.8)$$

the total differentials of the new variables are (keeping in mind they do not depend on higher derivatives y', y", etc)

$$dv = D_x v\, dx = \frac{\partial v}{\partial x}dx + \frac{\partial v}{\partial y}\frac{dy}{dx}dx \quad (1.5.9)$$
$$du = D_x u\, dx = \frac{\partial u}{\partial x}dx + \frac{\partial u}{\partial y}\frac{dy}{dx}dx \quad (1.5.10)$$

Clearly these total differentials, as well as the total derivative, only make sense along the curve v(u). Dividing, we get the same expression as above

$$\frac{dv}{du} = \frac{D_x v}{D_x u} = \frac{\frac{\partial v}{\partial x} + y'\frac{\partial v}{\partial y}}{\frac{\partial u}{\partial x} + y'\frac{\partial u}{\partial y}} = \frac{v_x + v_y y'}{u_x + u_y y'} \quad (1.5.11)$$

The condition for symmetry (1.5.2) , from which the symmetry itself can be derived, therefore becomes

$$\frac{v_x + \omega(x,y)\, v_y}{u_x + \omega(x,y)\, u_y} = \omega\,(u, v) \quad \textbf{(1.5.12)}$$

Later we will see why symmetries that map to the same curves (i.e. trivial symmetries) cannot be used. For now let's assume that we are dealing with non-trivial symmetries. We can start to see why such symmetries can be useful for solving ODEs if we assume an ODE admits the following specific symmetry:

$$(u, v) = (x, y + \epsilon) \quad (1.5.13)$$

Since we have *assumed* a symmetry we do not need to use (1.5.12) to find it. In this argument, however, we are not starting with a specific ODE to solve, i.e. we do not know ω explicitly. We can therefore use (1.5.12) "backwards", since we know it must in any case be true, to show in general what this kind of symmetry does:

$$\frac{0 + \omega(x,y)}{1 + 0} = \omega\,(x, y + \epsilon)$$
$$\omega\,(x, y) = \omega\,(x, y + \epsilon) \quad (1.5.14)$$

Now differentiate w.r.t. ϵ at $\epsilon = 0$ (note that $u \neq u(\epsilon)$, while $v = y + \epsilon$):

$$0 = \left(\frac{\partial \omega}{\partial u}\frac{du}{d\epsilon} + \frac{\partial \omega}{\partial v}\frac{dv}{d\epsilon}\right)_{\epsilon = 0}$$
$$0 = \omega_v|_0 \quad (1.5.15)$$

At $\epsilon = 0$, $v = y$ and $\omega_v = \omega_y$, so

$$\omega_y = 0 \quad (1.5.16)$$
$$\omega\,(x, y) = f(x) \quad (1.5.17)$$
$$\frac{dy}{dx} = f(x)$$
$$y(x) = \int f(x)\, dx + C \quad (1.5.18)$$

So this shows how we have reduced the problem to quadrature. We can make two points. The first is that the introduction of a continuous transformation allowed us to evaluate the derivative of the transformed equation (1.5.14) in the vicinity of the original equation (i.e. at $\epsilon = 0$), which led to the solution. The second is that if we can transform an ODE to one that admits (1.5.13) as a symmetry we have automatically solved the ODE. Thus, we are introducing the need for an additional transformation. Before we talk about this new transformation, let's define more precisely what we mean by "orbits", introducing some additional notation.

■ 6. Orbits

It is helpful to visualise the path followed by a point that belongs to a solution curve of an ODE as it is acted upon repeatedly by a non-trivial symmetry, each time as an infinitesimal step. As mentioned above, such a point follows what is called an "orbit" of the symmetry, each time jumping onto a new solution curve of the ODE. In fact, we can interpret the point (x, y) in Eq (1.4.3), belonging to a solution curve, as the starting point of a curve expressed in parametric form, where ϵ is the parameter:

$$u(\epsilon) \qquad v(\epsilon) \tag{1.6.1}$$

These parametric curves are the orbits of the symmetry, which are necessarily also invariant curves. We will see that the functions for the orbits can usually be integrated and found explicitly, but for the moment we work with the "local" point of view.

As shown in Fig 10, let's define the tangent vector to an orbit at the point (x, y) as

$$(\xi(x,\ y),\ \eta(x,\ y)) = \left(\left(\frac{du}{d\epsilon}\right)_0 ,\ \left(\frac{dv}{d\epsilon}\right)_0 \right) \tag{1.6.2}$$

With this we can write the Taylor series expansion about (x, y) for the Lie group action (truncated to 2 terms) as

$$u(x, y, \epsilon) = x + \epsilon\, \xi(x, y) \tag{1.6.3}$$

$$v(x, y, \epsilon) = y + \epsilon\, \eta(x, y) \tag{1.6.4}$$

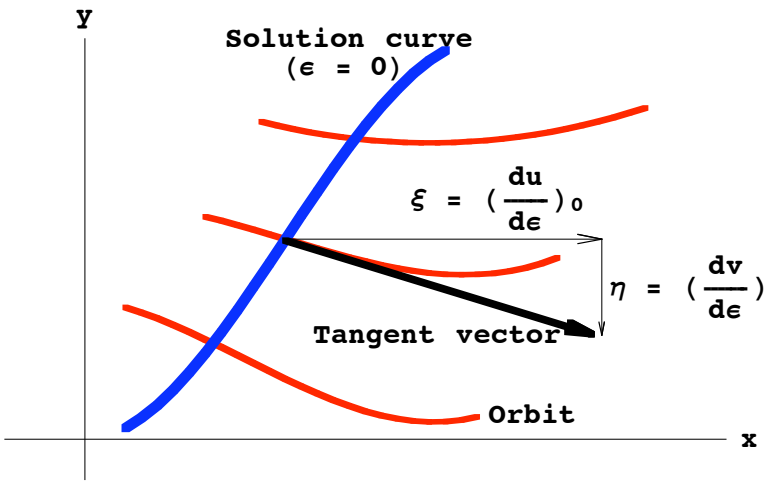


Fig 12. Orbits

For later use, we define the characteristic

$$Q(x, y, y') = \eta(x, y) - y'\, \xi(x, y) \tag{1.6.4}$$

And the reduced characteristic

$$\overline{Q}(x, y) = \eta(x, y) - \omega(x, y)\, \xi(x, y) \tag{1.6.5}$$

A solution is invariant and therefore the symmetry trivial if and only if

$$\overline{Q}(x, y) = 0 \qquad \text{on} \qquad y = f(x) \tag{1.6.6}$$

Sometimes invariant solutions can be used a bit like the fixed points in the qualitative analysis of differential equations, to determine the boundaries (topology) of regions where the symmetry can be used.

■ 7. Canonical coordinates

Now that we have an understanding of orbits we can introduce the additional transformation mentioned above. Given an ODE we want to visualise its solution curves and the orbits of any symmetry it might admit together, as a coordinate grid. We want to find a first transformation to "straighten" the grid so that we can then apply a symmetry of the type (1.5.13) and solve the ODE.

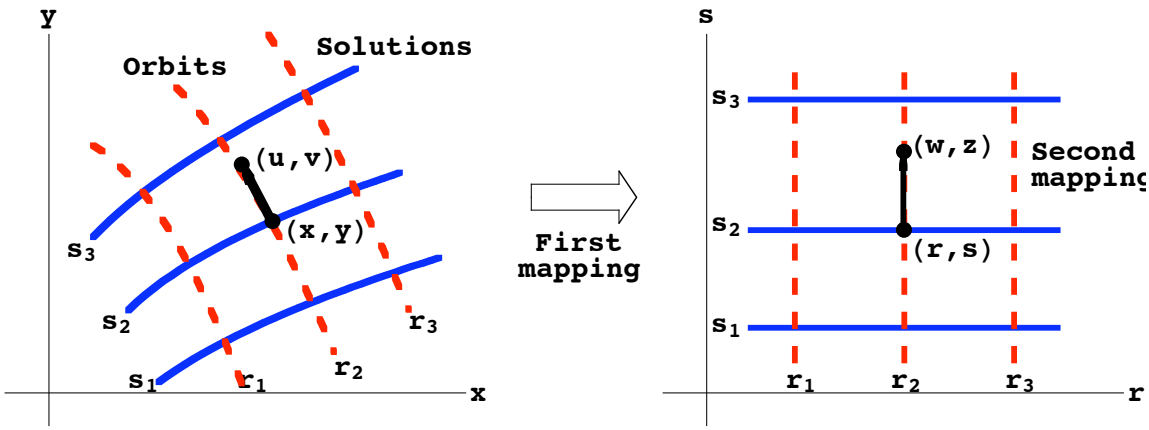


Fig 13. Canonical Coordinates

Symbolically we can draw the following:

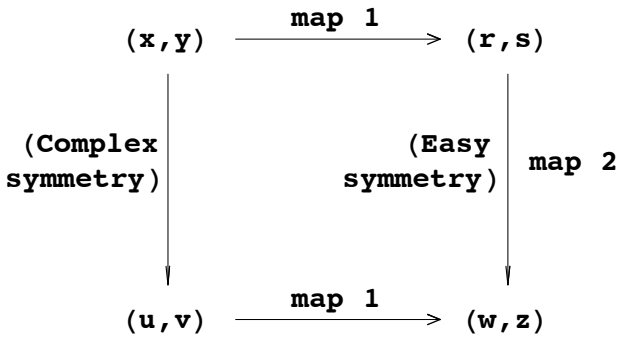


Fig 14. The Two Maps

The complex symmetry on the left is as we have discussed so far. Its explicit knowledge, by itself, does not necessarily help. However, it can be used to find Map 1 and greatly simplify the problem, as we now show.

We want to find functions r(x, y) and s(x, y) in terms of which the complex symmetry becomes

$$(w, z) = (r, s + \epsilon), \qquad (1.7.1)$$

because we saw that this leads to a trivial integration. Thus, we impose

$$\left(\frac{dr}{d\epsilon}\right)_0 = \frac{dr}{d\epsilon} = 0 \qquad \left(\frac{ds}{d\epsilon}\right)_0 = \frac{ds}{d\epsilon} = 1 \qquad (1.7.2)$$

By the chain rule,

$$\frac{dr}{d\epsilon} = \frac{\partial r}{\partial x}\frac{dx}{d\epsilon} + \frac{\partial r}{\partial y}\frac{dy}{d\epsilon} = \xi(x, y)\, r_x + \eta(x, y)\, r_y = 0 \quad (1.7.3)$$

$$\frac{ds}{d\epsilon} = \frac{\partial s}{\partial x}\frac{dx}{d\epsilon} + \frac{\partial s}{\partial y}\frac{dy}{d\epsilon} = \xi(x, y)\, s_x + \eta(x, y)\, s_y = 1 \quad (1.7.4)$$

We also require a non-zero Jacobian, which is equivalent to ensuring invertibility of Map 1:

$$r_x s_y - r_y s_x \neq 0 \quad (1.7.5)$$

This is because Eqs (1.7.3) and (1.7.4) can be re-written as

$$\begin{pmatrix} r_x & r_y \\ s_x & s_y \end{pmatrix}\begin{pmatrix} \xi \\ \eta \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \quad (1.7.6)$$

The non-zero Jacobian means that the determinant of the coefficient matrix is not zero, i.e. the matrix is not singular. Thus at any point where this is the case (ξ, η) can be found that satisfy Eq (1.7.6). This is not our prime concern because we are currently assuming that we know (ξ, η) and are looking for (r, s), but in any case it is good to know that the canonical coordinates and the tangent vector field to the orbits are not incompatible. More importantly, if the matrix is not singular it means that the column vectors are linearly independent and the row vectors are linearly independent. The latter are more meaningful because they represent the gradients of the two surfaces r(x, y) and s(x,y). Thus, if (1.7.5) applies no real number a exists such that

$$(r_x, r_y) = a\,(s_x, s_y), \quad (1.7.7)$$

meaning that the gradients are not parallel. If the gradients are not parallel then neither are the contours of the two surfaces, i.e. the invertibility condition ensures that the canonical coordinates always cross at a transversal angle and are never parallel. This all sounds good, but in the derivations to follow condition (1.7.5) is actually never used, so it is not clear exactly how invertibility can be assured.

Whereas the curves of constant-r are the orbits of the symmetry, it is not clear whether the curves of constant-s are the solutions.

Hydon makes the point that the solution to Eqs (1.7.3)-(1.7.4) is not unique, i.e. there is more than one set of canonical coordinates that satisfies them. For instance, any arbitrary smooth functions F(r) and G(r) can be added to (r, s) to obtain a new set of canonical coordinates, as follows:

$$(F(r), s + G(r)) \quad (1.7.8)$$

It is not clear if/when we will need this fact, but certainly (r, s) should be as simple as possible.

We can say a couple more interesting things. First, Eqs (1.7.3)-(1.7.4) show that the canonical coordinates are tightly dependent on the symmetry, through its tangent space. Because the symmetry, in turn, is derived from a knowledge of the ODE (see Eq (1.5.12)), canonical coordinates are closely related to the ODE and its invariant properties. Thus we discover that each ODE implies the existence of a great deal more structure; it is as if each ODE floated on an invisible "current" whose streamlines are the orbits of a symmetry. If we can express the ODE in terms of these streamlines and of a set of transversal curves the integration becomes trivial. This helps us see why the requirement of conformal invariance is useful.

Second, we know that r(x, y) is supposed to coincide with the orbits of the symmetry. We might therefore expect it to be related to the symmetry itself. This is precisely the case. r(x, y) denotes a surface defined on the x-y plane. The contours of this surface are lines of constant-r (by definition) and are therefore the orbits. The symmetry (1.4.3) (from whose tangent vector field r is derived) is none other than the same family of curves expressed in parametric form. So, eliminating ϵ from Eqs (1.4.3) leads to a function v(u) for a particular "starting point" (x, y) that corresponds to a particular orbit y(x, r) for a particular value of r.

Because we need to find also s(x, y), let's now formalise all this in terms of first integrals and the method of

characteristics. For any 1st-order ODE

$$\frac{dy}{dx} = f(x, y) \tag{1.7.9}$$

we can envision its solutions as contours of a function $\phi(x, y)$. $\phi(x, y)$ is called a first integral. Along its contours we know that, by definition,

$$d\phi = \frac{\partial \phi}{\partial x} dx + \frac{\partial \phi}{\partial y} dy = 0 \tag{1.7.10}$$

which we can re-write as

$$\phi_x + \frac{dy}{dx} \phi_y = 0 \tag{1.7.11}$$

But, since its contours are also solutions of (1.7.9), we can write:

$$\phi_x + f(x, y)\, \phi_y = 0 \tag{1.7.12}$$

If we now compare (1.7.12) to (1.7.3),

$$r_x + \frac{\eta(x,y)}{\xi(x,y)} r_y = 0 \tag{1.7.13}$$

we deduce that r(x, y) is a first integral of the ODE

$$\frac{dy}{dx} = \frac{\eta(x,y)}{\xi(x,y)} \tag{1.7.14}$$

This tells us that we can find r(x, y) either by eliminating ϵ in the symmetry or by solving (1.7.14). Notice that (1.7.14) is in general easier to solve than (1.5.1).

The method of characteristics is invoked to obtain s. First we notice that, since $\frac{ds}{d\epsilon} = 1$ (Eq (1.7.4)), s must change along the orbits the same way as ϵ. The method of characteristics leverages the contours of a surface to simplify a given PDE. In this case both r(x, y) and s(x, y) are surfaces and therefore their solution ought to involve a PDE. Their solution along orbits, however, allows us to cast the problem in terms of ODEs (such as (1.7.14)). So this would seem to imply that orbits are characteristic curves. This is not certain, nor is the precise relationship to Eqs (1.6.4)-(1.6.5) clear. In any case, we can say some useful things that let us solve for s. Referring to Fig 15,

$$ds = \frac{dx}{\frac{dx}{d\epsilon}} = \frac{dx}{\xi} \tag{1.7.15}$$

$$ds = \frac{dy}{\frac{dy}{d\epsilon}} = \frac{dy}{\eta} \tag{1.7.16}$$

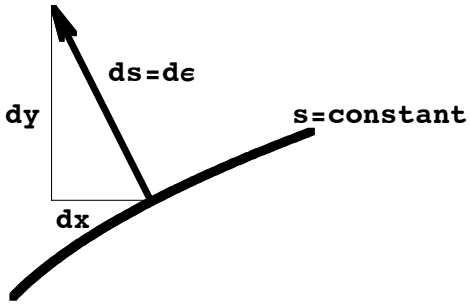


Fig 15. Orbit and Characteristic

Having found r(x, y) from (1.7.12), we invert this function and solve for y(x, r). Then we can use (1.7.15) to integrate:

$$s(x, r) = \int \frac{dx}{\xi(x, y(x, r))} \tag{1.7.17}$$

This integral is performed holding r constant. Finally,

$$s(x, y) = s(x, r(x, y)) \tag{1.7.18}$$

In cases where $\xi(x, y) = 0$, we can infer that the vector field is composed entirely of vertical vectors. Therefore, $r(x, y) = x$ automatically. From (1.7.16) we can then obtain s(x, y) as

$$s(x, y) = \int \frac{dy}{\eta(x, y)} \tag{1.7.19}$$

Once the canonical coordinates are found we apply them to transform the original ODE in terms of (r, s). This is easily done by following this simple argument in 4 steps:

1- The original differential equation in (x, y), Eq (1.5.1), can be seen as the definition of a 2D surface over the (x, y)-plane, indicating the value $\omega(x, y)$ that the derivative of the function y(x) takes at any point (x, y).
2- Given a map (x, y) ⟶ (u, v), for any function y(x) Eq (1.5.7) (or 1.5.11) tells us how the derivative of y(x) is mapped to the derivative of v(u)
3- But if in place of y'(x) we substitute $\omega(x, y)$, then Eq (1.5.7) tells us how the shape of this 2D surface gets changed by the mapping, i.e. how it appears over the u-v plane.
4- Applying Step 1 in reverse, it follows that (1.5.7) with $\omega(x, y)$ in place of y' is none other than the same differential equation in the new coordinates.

Now that we understand this well, let's follow Hydon's summary of why canonical coordinates are useful, which will be used also in Section 11 to show that canonical coordinates are equivalent to the integrating factor of elementary differential equations.

Hydon recapitulates that given the map (x, y) ⟶ (r, s) the original differential equation, using Eq (1.5.7), is mapped to the new coordinates (r, s):

$$\frac{ds}{dr} = \frac{s_x + s_y\ \omega(x(r, s), y(r, s))}{r_x + r_y\ \omega(x(r, s), y(r, s))} \tag{1.7.20}$$

$$\frac{ds}{dr} = \Omega(r, s) \tag{1.7.21}$$

Since the map that defines canonical coordinates is not in general a symmetry, (1.5.12) does not apply and the RHS is shown as $\Omega(r, s)$ rather than $\omega(r, s)$. However, since (r, s) are canonical coordinates, by definition we know that the new ODE is invariant under symmetries of the form (1.7.1), that we referred to earlier as "easy symmetries". From Eq (1.5.17) we know that under these circumstances the RHS is actually a function of r only:

$$\frac{ds}{dr} = \Omega(r) \tag{1.7.22}$$

Thus, as we advertised, the solution is given by (1.5.18), written in the form

$$s - \int \Omega(r)\, dr = C \tag{1.7.23}$$

yielding as the general solution to (1.5.1)

$$s(x, y) - \int^{r(x, y)} \Omega(r)\, dr = C \tag{1.7.24}$$

In short, to solve a 1st-order ODE first we find one of its symmetries, then we find the corresponding canonical coordinates, then the solution falls out as a simple integral (or "by quadrature").

■ 8. Linearised symmetry condition

By now we have outlined a method that leads to solutions in many cases, if not all. We have not addressed yet, however, how the symmetries are found in the first place. Eq (1.5.12) is the condition to be used, but in that form it is generally a non-linear PDE that's hard to solve, often harder than the original ODE. Here is where the advantage of Lie groups becomes evident. By defining continuous transformations we can explore their symmetry properties in the vicinity of the original solution--as before without worrying about the fact that we do not yet know what that solution is. So we can linearise Eq (1.5.12) and look for symmetries that hold for small ϵ; through integration we can usually also find the global map, although this is generally not necessary.

So we need to substitute the Taylor series expansion, Eqs (1.6.3)-(1.6.4), for the Lie symmetry into Eq (1.5.12). For convenience, let's first evaluate the partial derivatives:

$$u_x = 1 + \epsilon\, \xi_x \tag{1.8.1}$$
$$u_y = \epsilon\, \xi_y \tag{1.8.2}$$
$$v_x = \epsilon\, \eta_x \tag{1.8.3}$$
$$v_y = 1 + \epsilon\, \eta_y \tag{1.8.4}$$

In the following, we start using symbolic manipulation. The main equations are shown also in the text for better readability and to keep assigning equation numbers.

```
Clear[ξ, η, ω, u, v]
u[x_, y_, ϵ_] = x + ϵ ξ[x, y]
v[x_, y_, ϵ_] = y + ϵ η[x, y]
u_x = D[u[x, y, ϵ], x]
u_y = D[u[x, y, ϵ], y]
v_x = D[v[x, y, ϵ], x]
v_y = D[v[x, y, ϵ], y]
```

$x + \epsilon\, \xi[x, y]$

$y + \epsilon\, \eta[x, y]$

$1 + \epsilon\, \xi^{(1,0)}[x, y]$

$\epsilon\, \xi^{(0,1)}[x, y]$

$\epsilon\, \eta^{(1,0)}[x, y]$

$1 + \epsilon\, \eta^{(0,1)}[x, y]$

We enter the symmetry condition here, and separate the left- and right-hand sides:

```
SymmetryCond = (v_x + ω[x, y] v_y) / (u_x + ω[x, y] u_y) == ω[u[x, y, ϵ], v[x, y, ϵ]]
LHS = Part[SymmetryCond, 1]
RHS = Part[SymmetryCond, 2]
```

$$\frac{\omega[x, y]\,(1 + \epsilon\, \eta^{(0,1)}[x, y]) + \epsilon\, \eta^{(1,0)}[x, y]}{1 + \epsilon\, \omega[x, y]\, \xi^{(0,1)}[x, y] + \epsilon\, \xi^{(1,0)}[x, y]} == \omega[x + \epsilon\, \xi[x, y], y + \epsilon\, \eta[x, y]]$$

$$\frac{\omega[x, y]\,(1 + \epsilon\, \eta^{(0,1)}[x, y]) + \epsilon\, \eta^{(1,0)}[x, y]}{1 + \epsilon\, \omega[x, y]\, \xi^{(0,1)}[x, y] + \epsilon\, \xi^{(1,0)}[x, y]}$$

$\omega[x + \epsilon\, \xi[x, y], y + \epsilon\, \eta[x, y]]$

Now we need to derive the linearised symmetry condition, i.e. for small ϵ. This is done by taking the Taylor series expansion about $\epsilon = 0$ of the RHS and LHS of the symmetry condition, separately, truncating to retain only $O(\epsilon)$, and simplifying:

```
Clear[temp1, temp2]
temp1 = Normal[     Series[LHS, {ϵ, 0, 1}]     ]
temp2 = Normal[     Series[RHS, {ϵ, 0, 1}]     ]
```

```
ω[x, y] + ϵ (ω[x, y] η^(0,1)[x, y] + η^(1,0)[x, y] + ω[x, y] (-ω[x, y] ξ^(0,1)[x, y] - ξ^(1,0)[x, y]))
```

```
ω[x, y] + ϵ (η[x, y] ω^(0,1)[x, y] + ξ[x, y] ω^(1,0)[x, y])
```

The linearised symmetry condition for first-order ODEs therefore becomes:

```
LinSymCond = Part[ TraditionalForm[ Reduce[ temp1 == temp2 ] ], 1, 2]
```

```
-ω[x, y] η^(0,1)[x, y] + ω[x, y]^2 ξ^(0,1)[x, y] +
  η[x, y] ω^(0,1)[x, y] + ω[x, y] ξ^(1,0)[x, y] + ξ[x, y] ω^(1,0)[x, y] == η^(1,0)[x, y]
```

This equation can be rearranged as follows:

$$\eta_x + \omega\,(\eta_y - \xi_x) - \omega^2 \xi_y = \omega_x \xi + \omega_y \eta \qquad (1.8.5)$$

The linearised symmetry condition can be used to deduce/derive the tangent vector field components ξ and η given an ω (x, y), which is the RHS of a 1st-order ODE in standard form, Eq (1.5.1). This is usually done by assuming a functional form for ξ(x, y) and η(x, y). Whatever is allowed to vary is then determined by solving the resulting over-determined set of simple linear PDEs.

Following Hydon, the linearised symmetry condition can be re-written in terms of the reduced characteristic, Eq (1.6.5):

$$\begin{aligned}
&\overline{Q} = \eta - \omega\,\xi \\
&\eta_x - \omega\,\xi_x - \omega_x \xi + \omega\,\eta_y - \omega_y \eta - \omega^2 \xi_y = 0 \\
&\frac{\partial}{\partial x}(\eta - \omega\,\xi) + \omega\,\eta_y - \omega^2\,\xi_y - \omega\,\omega_y \xi = \omega_y \eta - \omega\,\omega_y \xi \\
&\frac{\partial}{\partial x}(\eta - \omega\,\xi) + \omega\,\frac{\partial}{\partial y}(\eta - \omega\,\xi) = \omega_y\,(\eta - \omega\,\xi) \\
&\overline{Q}_x + \omega\,\overline{Q}_y = \omega_y\,\overline{Q}
\end{aligned} \qquad (1.8.6)$$

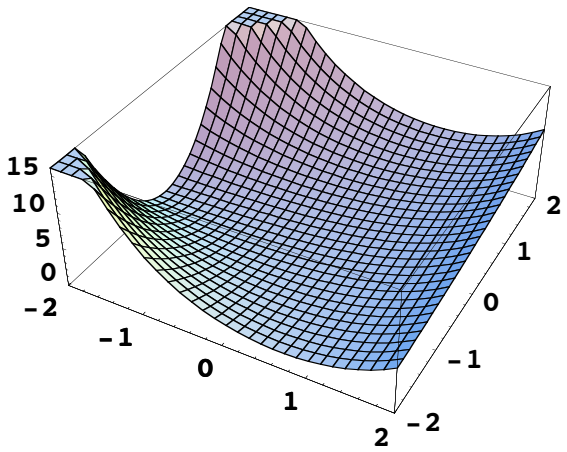
■ 9. A detailed example

Let's take one of Hydon's end-of-chapter problems and work all the way to its solution:

$$\frac{dy}{dx} = e^{-x} y^2 + y + e^x \qquad (1.9.1)$$

We might as well visualise ω(x, y) right away:

Fig 16. Plot of $\omega(x,y)$



We seek a symmetry that will leave the shape of the above surface invariant. To do so we assume the following functional form for ξ and η:

$$\xi(x, y) = a_1 + a_2 x + a_3 y \qquad (1.9.2)$$
$$\eta(x, y) = b_1 + b_2 x + b_3 y \qquad (1.9.3)$$

The symbolic manipulation of the subscripted variables in the following derivation has been made much easier and more convenient by the package SubscriptSymbols developed by Ted Ersek (copyright 2002, ted.ersek@navy.mil).

```
SubscriptSymbols[a, On]
SubscriptSymbols[b, On]
Clear[ξ, η, ω, temp, temp1, a₁, a₂, a₃, b₁, b₂, b₃];
ω[x_, y_] = Exp[-x] y^2 + y + Exp[x]
ξ[x_, y_] = a₁ + a₂ x + a₃ y
η[x_, y_] = b₁ + b₂ x + b₃ y
```

$$e^{x} + y + e^{-x} y^{2}$$

$$a_1 + a_2 x + a_3 y$$

$$b_1 + b_2 x + b_3 y$$

Now substitute into the linearised symmetry condition and solve for the coefficients of the tangent vector components:

```
LinSymCond
temp = Expand[Part[LinSymCond, 1]] - b₂
temp1 = Collect[temp, y]
Solve[Coefficient[temp1, y^4] == 0, a₃]
```

$$(b_1 + b_2 x + b_3 y)(1 + 2 e^{-x} y) + (a_1 + a_2 x + a_3 y)(e^{x} - e^{-x} y^{2}) + a_2 (e^{x} + y + e^{-x} y^{2}) - b_3 (e^{x} + y + e^{-x} y^{2}) + a_3 (e^{x} + y + e^{-x} y^{2})^{2} == b_2$$

$$b_1 - b_2 + a_1 e^{x} + a_2 e^{x} - b_3 e^{x} + a_3 e^{2x} + b_2 x + a_2 e^{x} x + a_2 y + 2 b_1 e^{-x} y + 3 a_3 e^{x} y + 2 b_2 e^{-x} x y + 3 a_3 y^{2} - a_1 e^{-x} y^{2} + a_2 e^{-x} y^{2} + b_3 e^{-x} y^{2} - a_2 e^{-x} x y^{2} + a_3 e^{-x} y^{3} + a_3 e^{-2x} y^{4}$$

$$b_1 - b_2 + a_1 e^{x} + a_2 e^{x} - b_3 e^{x} + a_3 e^{2x} + b_2 x + a_2 e^{x} x + (a_2 + 2 b_1 e^{-x} + 3 a_3 e^{x} + 2 b_2 e^{-x} x) y + (3 a_3 - a_1 e^{-x} + a_2 e^{-x} + b_3 e^{-x} - a_2 e^{-x} x) y^{2} + a_3 e^{-x} y^{3} + a_3 e^{-2x} y^{4}$$

$$\{\{a_3 \to 0\}\}$$

What we are doing here is set the coefficients of each power of x and y equal to zero since each power is orthogonal to all the others. Having found that $a_3 = 0$ we can set this value explicitly and then go on to the next power of y.

```
a₃ = 0;
temp1
Reduce[Coefficient[temp1, y^2] == 0]
```

```
b₁ - b₂ + a₁ ℯ^x + a₂ ℯ^x - b₃ ℯ^x + b₂ x + a₂ ℯ^x x +
  (a₂ + 2 b₁ ℯ^-x + 2 b₂ ℯ^-x x) y + (-a₁ ℯ^-x + a₂ ℯ^-x + b₃ ℯ^-x - a₂ ℯ^-x x) y^2
```

```
a₂ + b₃ - a₂ x == a₁
```

This tells us that $a_2 = 0$ since it's the only number multiplying x in this equation. So again we set it = 0 explicitly, deduce that $b_3 = a_1$, and go on to the next power:

```
a₂ = 0;
b₃ = a₁;
temp1 == 0
Reduce[Coefficient[temp1, y] == 0]
```

```
b₁ - b₂ + b₂ x + (2 b₁ ℯ^-x + 2 b₂ ℯ^-x x) y == 0
```

```
-b₂ x == b₁
```

b_2 is thus equal to zero, which means that also b_1 is equal to zero. This leaves only a_1 undetermined, so we may as well set $a_1 = 1$:

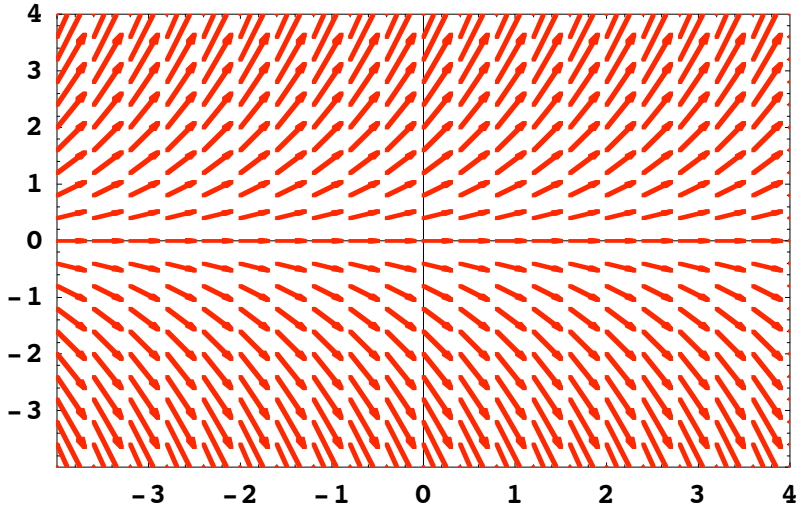
```
b₁ = 0;
b₂ = 0;
a₁ = 1;
ξ[x, y]
η[x, y]
```

```
1
```

```
y
```

Fig. 17 shows what this vector field looks like. It is not, unfortunately, very obvious why this vector field leaves the surface in Fig. 16 invariant. There may be other problems for which this is easier to see. Let's see whether by any chance this vector field has zero curl: $\frac{\partial y}{\partial x} - \frac{\partial 1}{\partial y} = 0$. So, Yes! Thus it can also be seen as the gradient of a scalar function, in particular f(x, y) = x + $y^2/2$. This does not seem to have any bearing at all on what we are doing now so we don't need to pursue the idea further for the moment.

Fig 17. Vector field for symmetry of Ex (1.9.1)



Now we can find the symmetry itself. Knowing that

$$\frac{du}{d\epsilon} = 1 \qquad \text{and} \qquad \frac{dv}{d\epsilon} = y \tag{1.9.4}$$

we infer that

$$u(x, y, \epsilon) = \epsilon + A(x, y) \tag{1.9.5}$$
$$v(x, y, \epsilon) = B(x, y)\, e^{\epsilon} \tag{1.9.6}$$

The initial conditions are

$$u(x, y, 0) = x \tag{1.9.7}$$
$$v(x, y, 0) = y \tag{1.9.8}$$

So that the symmetry is:

$$u(x, y, \epsilon) = x + \epsilon \tag{1.9.9}$$
$$v(x, y, \epsilon) = y\, e^{\epsilon} \tag{1.9.10}$$

Next, we need to find the canonical coordinates (r, s). Curves of constant-r are the orbits of the symmetry. Thus, the method of characteristics tells us that r(x, y) is a first integral of the ODE:

$$\frac{dy}{dx} = \frac{\eta(x, y)}{\xi(x, y)} = y \tag{1.9.11}$$

Leading to $y = C\, e^{x}$ or, (1.9.12)

$$r(x, y) = \frac{y}{e^{x}} \tag{1.9.13}$$

Now solve for y(x, r):

$$y(x, r) = r\, e^{x} \tag{1.9.14}$$

and substitute into:

$$s(x, r) = \int \frac{1}{\xi(x,\, y(x, r))}\, dx = \int 1\, dx = x + C \tag{1.9.15}$$

So,

$$(r, s) = \left(\frac{y}{e^{x}}, x\right) \tag{1.9.16}$$

Notice that r(x, y) is the surface whose contours are the orbits of the symmetry which, in turn, have already been expressed in parametric form through the symmetry itself given above as u(x, y, ϵ), v(x, y, ϵ), where ϵ is the

parameter. Fig. 18 shows that the orbits indeed look like the vector field of Fig. 17. Fig. 19 shows the same vector field plotted in 3D on the original surface $\omega(x, y)$.

Fig 18. Contours of r(x,y)

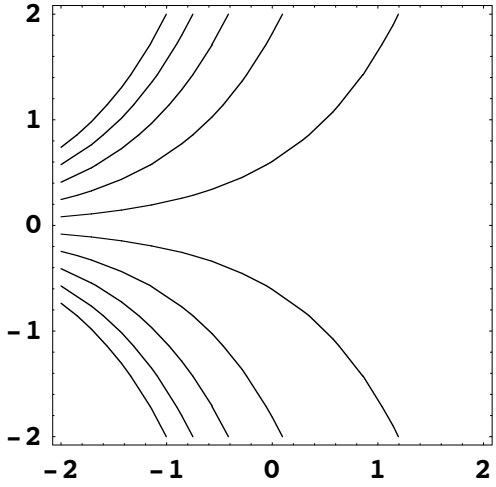
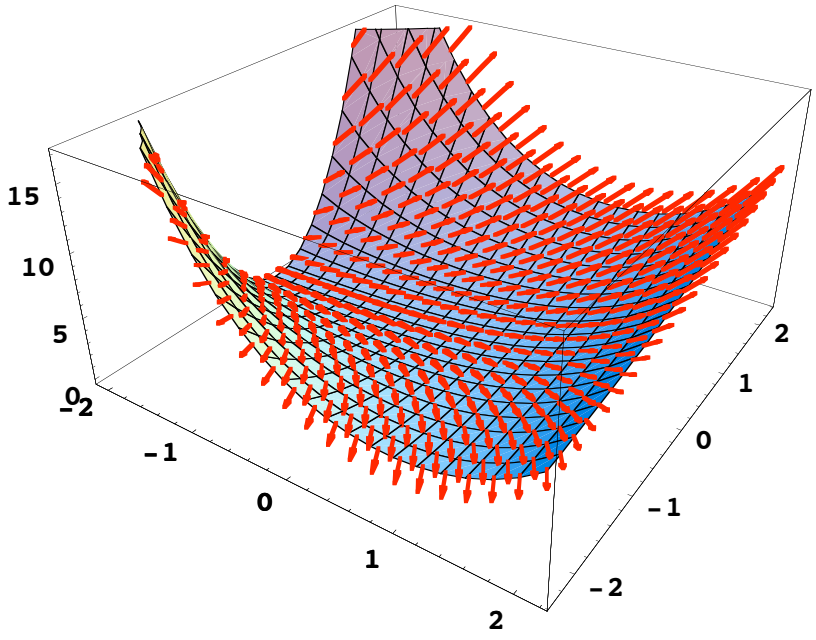


Fig 19. 3D vector field mapping ω(x,y) to itself



Having found the canonical coordinates now we need to transform the original ODE, Eq. (1.9.1), from (x, y) into (r, s) coordinates. For this we refer to the discussion in Section 7, but repeat the basic steps here. For later use let's invert the map:

$$x(r, s) = s \qquad (1.9.17)$$

$$y(r, s) = r\, e^{s} \qquad (1.9.18)$$

We need to remember that a map from (x, y) to (r, s) has, in itself, nothing what-so-ever to do with a particular function defined in either the x-y or the r-s planes. However, knowing the transformation T: [r(x, y), s(x, y)] can tell us how the derivative $\frac{dy}{dx}$ of any arbitrary function y(x) transforms into the corresponding derivative $\frac{ds}{dr}$. This is actually none other than the derivation we already went through to obtain Eq (1.5.7)

$$ds = \frac{\partial s}{\partial x}dx + \frac{\partial s}{\partial y}dy \tag{1.9.19}$$
$$dr = \frac{\partial r}{\partial x}dx + \frac{\partial r}{\partial y}dy \tag{1.9.20}$$

Dividing the first equation by the second, we obtain

$$\frac{ds}{dr} = \frac{s_x\,dx + s_y\,dy}{r_x\,dx + r_y\,dy} = \frac{s_x + s_y\,\frac{dy}{dx}}{r_x + r_y\,\frac{dy}{dx}} \tag{1.9.21}$$

Now we can do two different things: first, as in Eq (1.7.23), we can substitute ω (x, y) where $\frac{dy}{dx}$ appears, and use Eqs (1.9.17-18) to express everything in terms of (r, s) to obtain the ODE in the r-s plane:

$$\frac{ds}{dr} = \frac{s_x + s_y\;\omega(x(r,s), y(r,s))}{r_x + r_y\;\omega(x(r,s), y(r,s))} \tag{1.9.22}$$

$$\frac{ds}{dr} = \frac{1+0}{-y\,e^{-x} + e^{-x}\,[e^{-x}\,y^2 + y + e^x]}$$
$$\frac{ds}{dr} = \frac{1}{-r\,e^s\,e^{-s} + e^{-s}\,[e^{-s}\,r^2\,e^{2s} + r\,e^s + e^s]}$$
$$\frac{ds}{dr} = \frac{1}{-r + e^{-s}\,[r^2\,e^s + r\,e^s + e^s]}$$
$$\frac{ds}{dr} = \frac{1}{r^2 + 1} \tag{1.9.23}$$

Otherwise, we can use the same procedure to determine $\frac{dy}{dx}$ and, together with ω (x(r, s), y(r, s)), plug both into Eq (1.9.1). The result will be the same.

$$dy = \frac{\partial y}{\partial r}dr + \frac{\partial r}{\partial s}ds \tag{1.9.24}$$
$$dx = \frac{\partial x}{\partial r}dr + \frac{\partial x}{\partial s}ds \tag{1.9.25}$$
$$\frac{dy}{dx} = \frac{y_r\,dr + y_s\,ds}{x_r\,dr + x_s\,ds} = \frac{y_r + y_s\,\frac{ds}{dr}}{x_r + x_s\,\frac{ds}{dr}} = \frac{e^s + r\,e^s\,\frac{ds}{dr}}{0 + \frac{ds}{dr}} \tag{1.9.26}$$

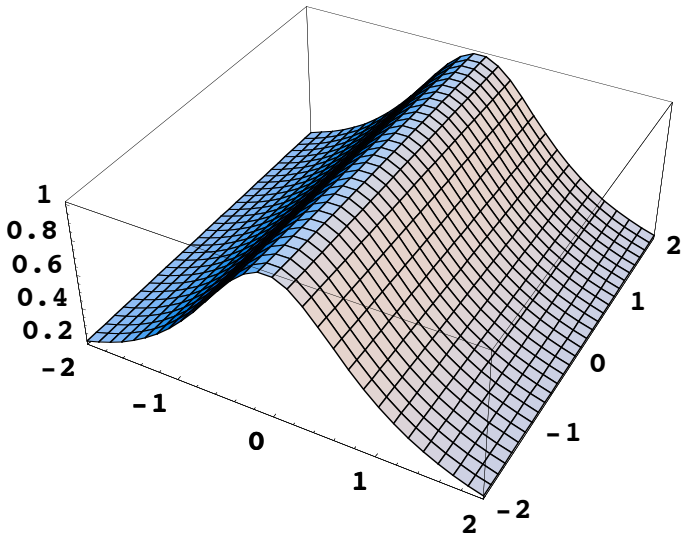
So,

$$\frac{e^s + r\,e^s\,\frac{ds}{dr}}{\frac{ds}{dr}} = e^{-x}y^2 + y + e^x$$

$$\frac{ds}{dr}[e^{-s}\,r^2\,e^{2s} + r\,e^s + e^s] = e^s + r\,e^s\,\frac{ds}{dr}$$
$$\frac{ds}{dr}[r^2 + 1] = 1$$
$$\frac{ds}{dr} = \frac{1}{r^2 + 1}$$

The not so surprising conclusion we can derive from this is that the map (x, y) ⟶ (r, s) is invertible. Note that, as Eq (1.5.17) had told us, the RHS of (1.9.23) is a f(r) only. Let's plot it:

Fig 20. Plot of Ω(r)



Integrating,

$$s(r) = \tan^{-1}(r) + c \qquad (1.9.27)$$

Substituting back from Eq (9.16),

$$x = \tan^{-1}(y\, e^{-x}) + c \qquad (1.9.28)$$

$$y\, e^{-x} = \tan(x + c) \qquad (1.9.29)$$

So the general solution to the ODE (1.9.1) is

$$\mathbf{y(x) = e^{x} \tan(x + c)} \qquad \mathbf{(1.9.30)}$$

Now that we have found the solution, let's see what it looks like. Let's also see how the solution is mapped by the symmetry into a new curve.

```
Clear[x, y, u, v, curve, curve1, curve2, x0, y0, x1, y1, vec, eps]
eps = .3;
y[x_] := Exp[x] * Tan[x]
curve1 := Line[Table[{i, y[i]}, {i, -1.5, 1.4, .02}]]
curve2 := Line[Table[{i, Exp[eps]*y[i - eps]}, {i, -1.2, 1.6, .02}]]
```

The above calculation is quite straightforward. Starting with the solution (curve1),

$$y = e^{x} \tan(x) \qquad (1.9.31)$$

substitute for x and y from Eqs. (1.9.9) and (1.9.10), respectively. This gives the function in terms of the new variables (u, v):

$$v\, e^{-\epsilon} = e^{u-\epsilon} \tan(u - \epsilon) \qquad (1.9.32)$$

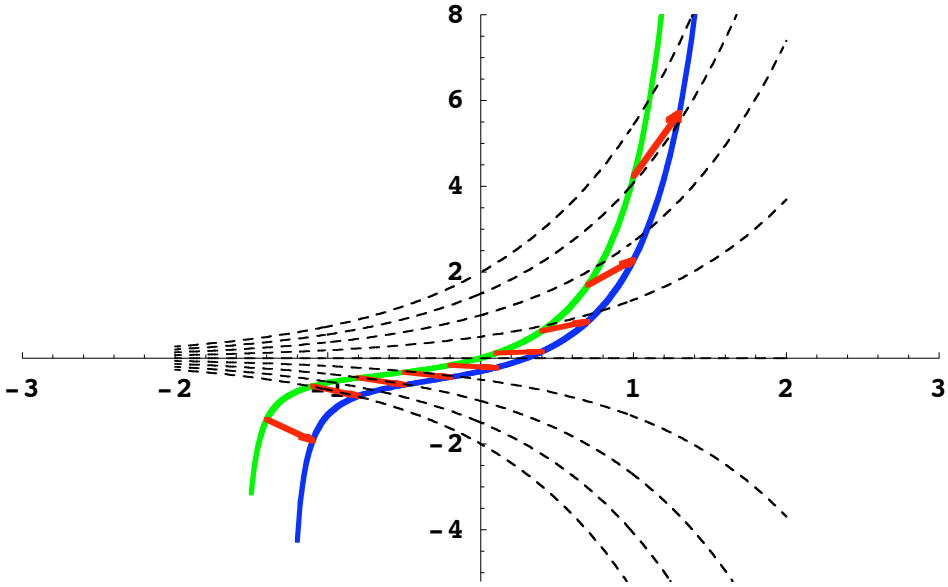
Now we can just rename (u, v) = (x, y) for convenience, since we plot both functions on the same axes, and solve for y to obtain curve2:

$$y = e^{\epsilon}\, [e^{x-\epsilon} \tan(x - \epsilon)]$$

$$y = e^{x} \tan(x - \epsilon) \qquad (1.9.33)$$

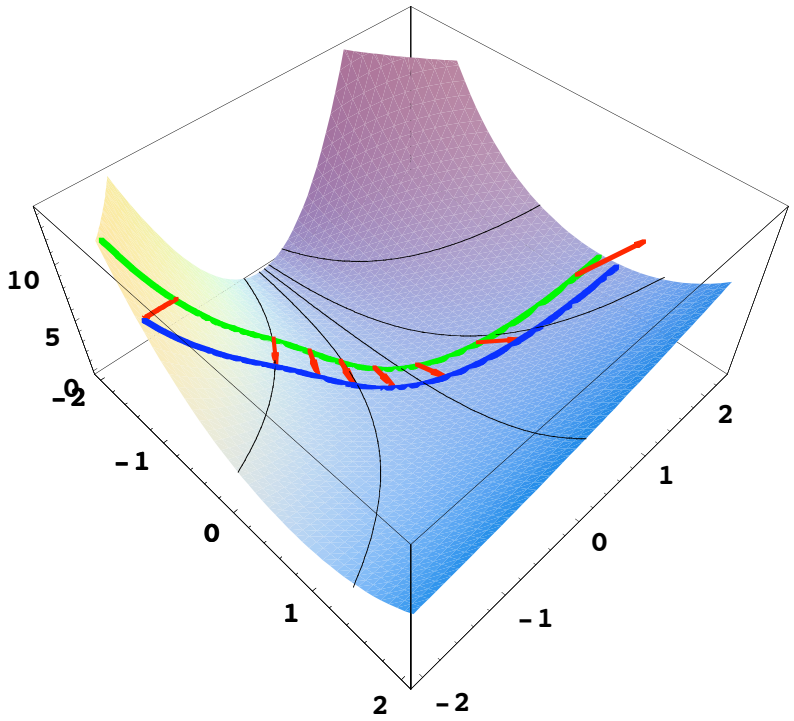
We also would like to see the orbits on the same graph, as shown in Fig. 21.

Fig 21. Action of the symmetry on solution curve



This shows how the solution "flows" along the orbits of the symmetry towards other solutions of the same ODE. Finally, we can also show the mapping of the solution curve in the space defined by the surface $\omega(x, y)$, where we can finally see why the symmetry condition ensures that solutions are mapped into solutions, since they lie on the same surface. The space defined by $\omega(x, y)$ is called the "jet space". The curves drawn on the surface in this and subsequent figures are made possible by the package CurveOnSurface developed by Gianluca Gorni at the University of Udine (www.dimi.uniud.it/~gorni/).

Fig 22. Mapping in the jet space



It may appear that we have "beaten this problem to death", but in fact there is still one graph that is of interest, the graph of the first integral, whose contours are the solution curves themselves! To plot it we use Eq. (1.9.28) to solve for c and set it equal to $\Phi(x, y)$:

$$\Phi(x, y) = x - \tan^{-1}(y\, e^{-x}) \qquad (1.9.34)$$

This is the light blue surface shown in Fig. 23. The two curves are two contours of Φ and therefore the same two solutions we just plotted. To see how they are related to each other let's investigate the effect of the symmetry on Φ. Let $\Phi = c_1$ be the first solution, i.e. the green one. Its equation is given by

$$c_1 = x - \tan^{-1}(y\, e^{-x}) \qquad (1.9.35)$$

Invert (1.9.9) and (1.9.10) to obtain x(u, v) and y(u, v), then substitute them into (1.9.35):

$$x = u - \epsilon \qquad (1.9.36)$$
$$y = v\, e^{-\epsilon} \qquad (1.9.37)$$
$$c_1 = u - \epsilon - \tan^{-1}(v\, e^{-\epsilon}\, e^{-(u-\epsilon)})$$
$$c_1 + \epsilon = u - \tan^{-1}(v\, e^{-u})$$

Reverting u → x and v → y we get

$$c_2 = x - \tan^{-1}(y\, e^{-x}) \qquad (1.9.38)$$

which is another solution, i.e. the blue curve, where $c_2 = c_1 + \epsilon$. This shows the exact meaning of the statement "symmetries map solutions into solutions". Fig. 24 finally shows the contour set in 3D.

Fig 23. First integral

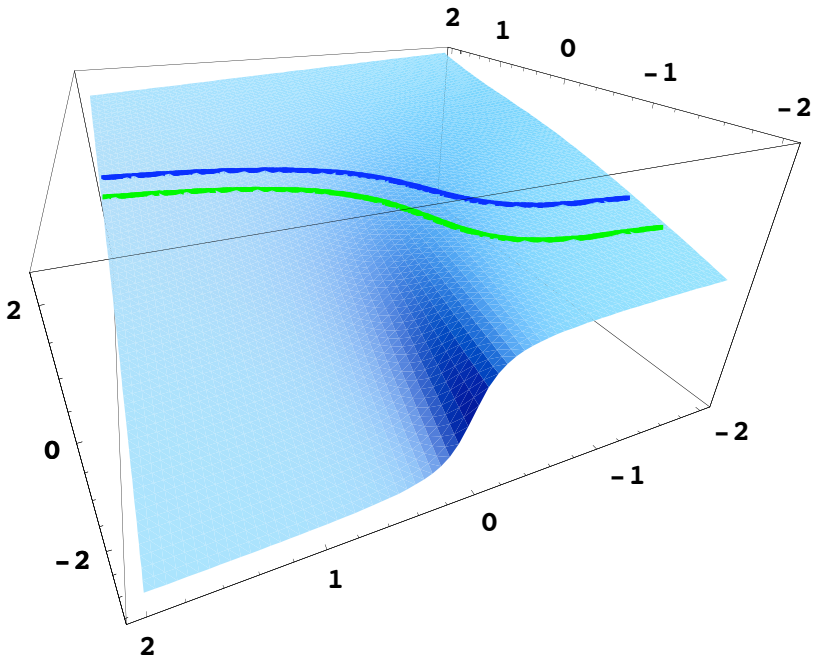
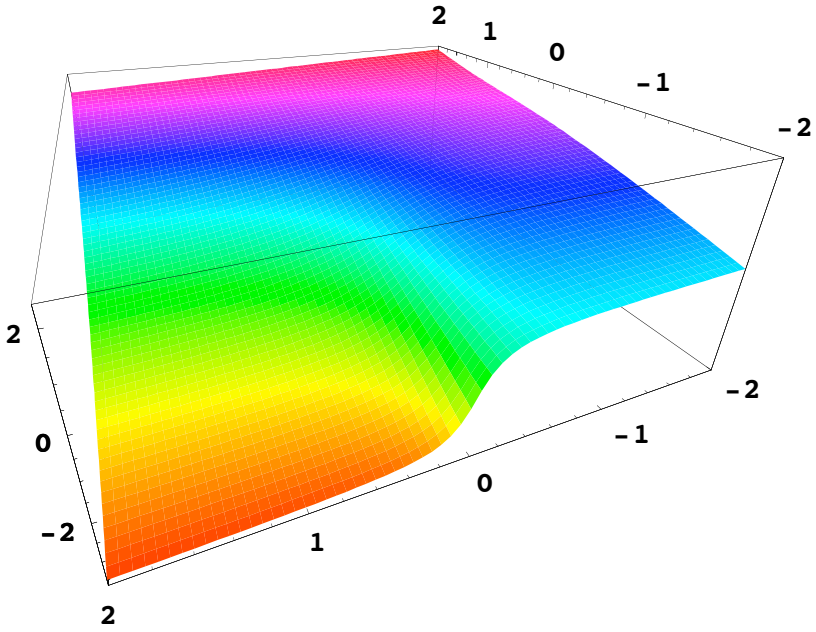


Fig 24. Contours of First Integral



■ 10. Symmetries and standard methods

Hydon now shows how this approach can be used to justify standard methods used to solve elementary linear differential equations. The general idea is that all first-order equations admitting a particular group of symmetries can be solved exactly using the corresponding canonical coordinates. Methods that apply to whole classes of ODEs therefore result. For example, take the general linear ODE:

$$y' + F(x)\, y = G(x) \qquad (1.10.1)$$

The corresponding homogeneous equation is

$$y_0' + F(x)\, y_0 = 0 \qquad (1.10.2)$$

with solution

$$y_0(x) = e^{-\int F(x)\, dx} \qquad (1.10.3)$$

Now we know that, if y(x) is a solution, so is $y(x) + y_0(x)$ and, more generally, $y(x) + \epsilon\, y_0(x)$. Since symmetries map solutions into solutions, this is equivalent to saying that the ODE (1.10.1) admits the symmetry

$$(u, v) = (x, y + \epsilon\, y_0(x)) \qquad (1.10.4)$$

By definition (see Eqs (1.6.3-4)), the vector field associated with this symmetry is

$$\xi(x, y) = 0 \qquad (1.10.5)$$
$$\eta(x, y) = y_0(x) \qquad (1.10.6)$$

From Eq (1.7.19) we know that the canonical coordinates are

$$r(x, y) = x \qquad (1.10.7)$$
$$s(x, y) = \int \frac{dy}{\eta(x, y)} = \int \frac{dy}{y_0(x)} = \frac{y}{y_0(x)} \qquad (1.10.8)$$

Now we can transform the ODE (1.10.1). First re-write it in the form of (1.5.1):

$$\frac{dy}{dx} = G(x) - F(x)\,y \qquad (1.10.9)$$

i.e.

$$\omega(x, y) = G(x) - F(x)\,y \qquad (1.10.10)$$

Then use (1.9.22) to transform:

$$\begin{aligned}\frac{ds}{dr} &= \frac{s_x + s_y\,\omega(x(r,s), y(r,s))}{r_x + r_y\,\omega(x(r,s), y(r,s))} \\ &= \frac{-\frac{y}{y_0^2(x)}\,y'_0(x) + \frac{1}{y_0(x)}[G(x) - F(x)\,y]}{1+0} \\ &= -\frac{y_0(r)\,s}{y_0^2(r)}\,y'_0(r) + \frac{1}{y_0(r)}[G(r) - F(r)\,y_0(r)\,s] \\ &= -\frac{s}{y_0(r)}\,y'_0(r) + \frac{1}{y_0(r)}\,G(r) - F(r)\,s \\ &= \frac{G(r)}{y_0(r)} - \frac{s}{y_0(r)}[y'_0(r) + F(r)\,y(r)]\end{aligned} \qquad (1.10.11)$$

By (1.10.2) we recognise that the term in square brackets is just the homogenous ODE, leaving

$$\frac{ds}{dr} = \frac{G(r)}{y_0(r)} \qquad (1.10.12)$$

Hence,

$$s(r) = \int \frac{G(r)}{y_0(r)}\,dr + C \qquad (1.10.13)$$

From which the general solution becomes

$$\frac{y(x)}{y_0(x)} = \int_0^x \frac{G(r)}{y_0(r)}\,dr + C \qquad (1.10.14)$$

Or, in the more familiar form of homogeneous plus particular solution,

$$y(x) = C y_0(x) + y_0(x)\int_0^x \frac{G(r)}{y_0(r)}\,dr \qquad (1.10.15)$$

Another method that is useful since it can be used as an alternative to the canonical coordinates is based on the integrating factor. This is discussed in the next section.

■ 11. Integrating factor

Hydon singles out the integrating factor as particularly useful since it is equivalent to using canonical coordinates. This will take a little work to show, mainly because the integrating factor derivation connects to a (very nice) theorem of vector calculus. Given the interpretation of symmetries as vector fields it is worth our while to refresh our memory about these elegant, albeit elementary, results.

As in the previous section, take the simple linear ODE in the form

$$\frac{dy}{dx} + F(x)\,y = G(x) \qquad (1.11.1)$$

An integrating factor is a function $\mu(x)$ that will allow the LHS to become a total or exact differential:

$$\mu(x)\,\frac{dy}{dx} + \mu(x)\,F(x)\,y = \frac{d}{dx}(\mu\,y) \qquad (1.11.2)$$

Another way to write this is

$$dh = \mu(x)\,dy + \mu(x)\,F(x)\,y\,dx \qquad (1.11.3)$$

where $h(x, y) = \mu(x)\,y$. Now, using the chain rule we also know that we can express an exact differential as

$$dh = \frac{\partial h}{\partial x} dx + \frac{\partial h}{\partial y} dy \quad (1.11.4)$$

which means that

$$\frac{\partial h}{\partial x} = \mu(x)\, F(x)\, y \quad (1.11.5)$$
$$\frac{\partial h}{\partial y} = \mu(x) \quad (1.11.6)$$

Now let's pause for a second and think about what an exact differential is. Take the surface h(x, y). The exact differential dh between two points (x_1, y_1) and (x_2, y_2) is simply the change in height of the surface h between these two points. This change in height can be measured by means of a line integral, as follows:

$$h_2 - h_1 = \int_1^2 dh \quad (1.11.7)$$

$$= \int_1^2 \left[\frac{\partial h}{\partial x} dx + \frac{\partial h}{\partial y} dy \right] = C \quad (1.11.8)$$

If these two points are fixed, since h is given and is also fixed, the difference in height dh will be constant, it will not vary regardless of which path one follows from the first to the second point. Another way to say this is that the line integral of an exact differential between two fixed end-points is constant. This simple fact has remarkable consequences.

First, if we retrace a path in the opposite direction, it need not be the same path, the result will be -dh. Thus, any closed path integral of an exact differential will vanish. We need to make sure that the region is simply-connected and that it does not contain singularities, but for the moment these are technical points that we don't need to worry about. Our intermediate result thus is

$$\oint dh = 0 \quad (1.11.9)$$

Now we recall one of the fundamental theorems of vector calculus, Stokes's theorem. It says that the integral of the curl of a vector field over an area is equal to the counter-clockwise closed path integral along the perimeter of that same area. This sounds rather abstract, but there is a nice interpretation from fluid mechanics that may help: to visualise the curl of a vector field imagine water flowing slowly in a shallow creek, so we can imagine the water as a 2-D sheet flowing smoothly. The water is following curved streamlines that may reflect the curved banks or perhaps the presence of obstacles in the creek itself. Now imagine a thin square piece of cork floating on the water. As the cork flows downstream carried by the water it may or may not spin around its vertical axis. If the cork is following a curved streamline and it changes its orientation w.r.t. the banks then the curl of the velocity field of the water (at the location of the cork) is not zero. If the cork maintains a fixed orientation with respect to the banks in spite of the fact that it may be following a curved streamline then the curl is zero. Never mind how likely or unlikely either of these two scenarios may appear; both are actually possible and even observed routinely. The curl of the velocity field of a fluid is a local property and is roughly equivalent to its vorticity, and the closed path integral around a patch of water is called the circulation. So the theorem says that the integral of the vorticity over a given "sheet" of water is equal to the net circulation around that sheet. If the vorticity is zero the circulation is zero. In symbols,

$$\iint \bar{\nabla} \times \bar{V} = \oint \bar{V} \cdot d\bar{s} \quad (1.11.10)$$

The over-bar indicates vector character. Now look again at Eq (1.11.8). It can also be written as the dot product between the gradient of the function h(x,y) and the directed line segment ds, also called the differential arc-length:

$$\frac{\partial h}{\partial x} dx + \frac{\partial h}{\partial y} dy = \bar{\nabla} h \cdot d\bar{s} \quad (1.11.11)$$

where

$$\bar{\nabla} h = u\, \hat{i} + v\, \hat{j} \quad (1.11.12)$$
$$d\bar{s} = dx\, \hat{i} + dy\, \hat{j} \quad (1.11.13)$$

and

$$u = \frac{\partial h}{\partial x} = \mu(x)\, F(x)\, y \quad (1.11.14)$$

$$v = \frac{\partial h}{\partial y} = \mu(x) \qquad (1.11.15)$$

Thus, Eq (1.11.9) is telling us that

$$\oint \bar{\nabla} h \cdot d\bar{s} = 0 \qquad (1.11.16)$$

By Stokes's theorem therefore we can say that

$$\oint \bar{\nabla} h \cdot d\bar{s} = \iint \bar{\nabla} \times \bar{\nabla} h = 0 \qquad (1.11.17)$$

Since this result applies regardless of the size of the patch of water, we can let it shrink to zero, obtaining the general result that the curl of the gradient of a scalar function is always identically zero:

$$\bar{\nabla} \times \bar{\nabla} h = 0 \qquad (1.11.18)$$

In 2-D, this means that

$$\frac{\partial v}{\partial x} - \frac{\partial u}{\partial y} = 0 \qquad (1.11.19)$$

Substituting for u and v,

$$\frac{\partial \mu(x)}{\partial x} - \frac{\partial}{\partial y}[\mu(x)\, F(x)\, y] = 0 \qquad (1.11.20)$$

Since $\mu = \mu(x)$ only,

$$\frac{d\mu}{dx} = \mu F \qquad (1.11.21)$$

Yielding finally for the integrating factor,

$$\mu(x) = e^{\int F(x)\, dx}, \qquad (1.11.22)$$

which is the well-known result from elementary differential equations. We could have said this by inspection, of course, but this more general machinery now allows us to analyse more complex ODEs. For instance, following Hydon, take Eq (1.5.1) written in Pfaffian form

$$dy - \omega(x, y)\, dx = 0 \qquad (1.11.23)$$

As before, we want to find an integrating factor $\mu(x, y)$, which is now a function of x and y, such that

$$\mu(x, y)\, dy - \mu(x, y)\, \omega(x, y)\, dx \qquad (1.11.24)$$

is an exact differential. As before we can write

$$\Phi_2 - \Phi_1 = \int [- \mu(x, y)\, \omega(x, y)\, dx + \mu(x, y)\, dy] = C \qquad (1.11.25)$$

$$\frac{\partial \Phi}{\partial x} = - \mu(x, y)\, \omega(x, y) \qquad (1.11.26)$$

$$\frac{\partial \Phi}{\partial y} = \mu(x, y) \qquad (1.11.27)$$

$$\bar{\nabla}\Phi = [- \mu(x, y)\, \omega(x, y)]\, \hat{i} + [\, \mu(x, y)]\, \hat{j} \qquad (1.11.28)$$

Applying zero curl again,

$$\frac{\partial}{\partial x}\mu + \frac{\partial}{\partial y}(\mu\, \omega) = 0 \qquad (1.11.29)$$

Now we need to make a bit of a jump to an earlier derivation. The above equation is in the same form as Eq

(1.8.6),

$$\overline{Q}_x + \omega \overline{Q}_y - \omega_y \overline{Q} = 0 \qquad (1.11.30)$$

where in this case the overbar does *not* indicate vector character, it indicates what Hydon calls the reduced characteristic. Hydon probably saw this by inspection because he sets

$$\mu(x, y) = \frac{1}{\overline{Q}} \qquad (1.11.31)$$

Plugging into (1.11.29),

$$\frac{\partial}{\partial x}\left(\frac{1}{\overline{Q}}\right) + \frac{\partial}{\partial y}\left(\frac{1}{\overline{Q}}\,\omega\right) = 0 \qquad (1.11.32)$$
$$-\frac{1}{\overline{Q}^2}\frac{\partial \overline{Q}}{\partial x} + \omega_y \frac{1}{\overline{Q}} - \frac{1}{\overline{Q}^2}\omega \frac{\partial \overline{Q}}{\partial y} = 0$$
$$-\overline{Q}_x + \omega_y \overline{Q} - \omega \overline{Q}_y = 0$$
$$\overline{Q}_x + \omega \overline{Q}_y - \omega_y \overline{Q} = 0$$

So, indeed, by a circuitous route we have found this more general integrating factor which, in general, equals the inverse of the reduced characteristic:

$$\mu(x, y) = \frac{1}{\overline{Q}} = \frac{1}{\eta - \omega \xi} \qquad (1.11.33)$$

and the solution, in implicit form, is

$$\int \frac{dy - \omega\, dx}{\eta - \omega \xi} = C \qquad \textbf{(1.11.34)}$$

Here Hydon shows that the integrating factor method is equivalent to finding canonical coordinates. From (1.7.20) and (1.7.22) we know that, if (r, s) are canonical coordinates,

$$\Omega(r) = \frac{s_x + \omega s_y}{r_x + \omega r_y} \qquad (1.11.35)$$

Thus we can solve for ω as

$$\omega = -\frac{s_x - \Omega(r) r_x}{s_y - \Omega(r) r_y} \qquad (1.11.36)$$

Now we recall Eqs (1.7.3) and (1.7.4):

$$\xi r_x + \eta r_y = 0 \qquad (1.7.3)$$
$$\xi s_x + \eta s_y = 1 \qquad (1.7.4)$$

Since (1.7.3) is equal to zero, we can add it to (1.7.4) without changing the result, and then rearrange:

$$\xi s_x - \Omega(r)[\xi r_x + \eta r_y] + \eta s_y = 1 \qquad (1.11.37)$$
$$\xi [s_x - \Omega(r) r_x] + \eta [s_y - \Omega(r) r_y] = 1 \qquad (1.11.38)$$

Using (1.11.36) twice in (1.11.38),

$$-\omega \xi [s_y - \Omega(r) r_y] + \eta [s_y - \Omega(r) r_y] = 1 \qquad (1.11.39)$$
$$\xi [s_x - \Omega(r) r_x] - \frac{\eta}{\omega} [s_x - \Omega(r) r_x] = 1 \qquad (1.11.40)$$

From this we deduce that

$$s_y - \Omega(r) r_y = \frac{1}{\eta - \omega \xi} \qquad (1.11.41)$$
$$s_x - \Omega(r) r_x = -\frac{\omega}{\eta - \omega \xi} \qquad (1.11.42)$$

At this point we can take the general solution in terms of the integrating factor, (1.11.34), and substitute Eqs (1.11.41) and (1.11.42):

$$\int [s_y - \Omega(r)\, r_y]\, dy + [s_x - \Omega(r)\, r_x]\, dx = C \qquad (1.11.43)$$

rearranging, we find the exact differentials for ds and dr:

$$\int [s_x\, dx + s_y\, dy] - \Omega(r)\, [r_x\, dx + r_y\, dy] = C \qquad (1.11.44)$$

$$\int [ds - \Omega(r)]\, dr = C \qquad (1.11.45)$$

Since this integral equals a constant, we deduce that the integrand is also an exact differential, dΦ. From this we recover the general solution (1.7.24):

$$\Phi(r, s) = s - \int \Omega(r)\, dr \qquad (1.11.46)$$

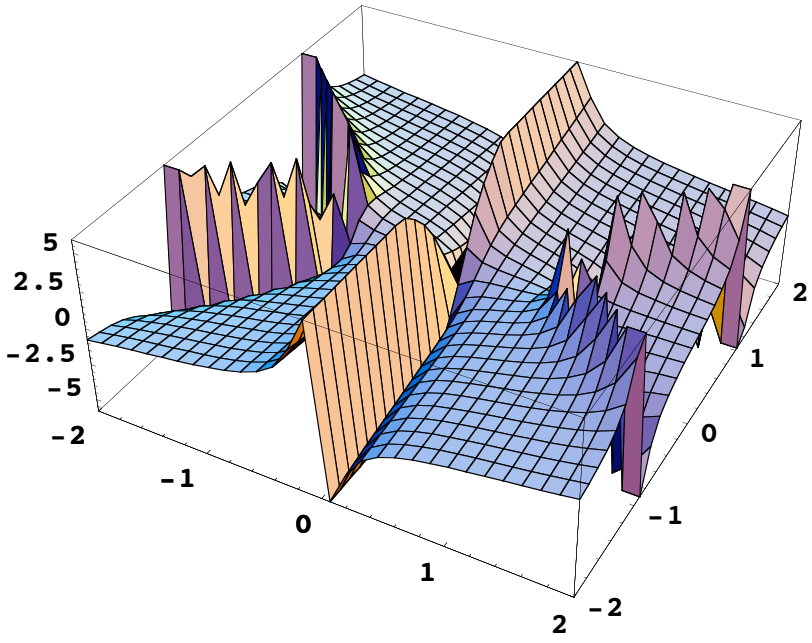
Which of course means that each solution expressed as s(r) is a contour of the 2-D surface Φ(r, s).

To conclude this section let's look at another example from Hydon.

$$\frac{dy}{dx} = \frac{y^3 + y - 3x^2 y}{3x y^2 + x - x^3} \qquad (1.11.47)$$

The surface $\omega(x, y)$ looks at bit pathological, as shown below.

Fig 25. Another $\omega(x,y)$



This ODE is apparently invariant under a symmetry group whose vector field, shown in Fig 23, is

$$(\xi, \eta) = (y^3 + y - 3x^2 y,\ x^3 - x - 3x y^2) \qquad (1.11.48)$$

and whose curl is not zero. The characteristic equation (1.7.14) for this problem is as hard as the original ODE. Using (1.11.34), on the other hand, yields an integrable function. After some simplification the final solution is given by

$$\frac{(y^2 + x^2)^2 + y^2 - x^2}{xy} = C \qquad (1.11.49)$$

Fig 24 shows the 2-D surface defined by this function, which is also called the first integral of the ODE, Φ(x, y). Fig 25 shows various contours of the first integral. The solution that we would normally seek corresponds to Φ = 0.

Fig 26. Visualisation of 2D vector field

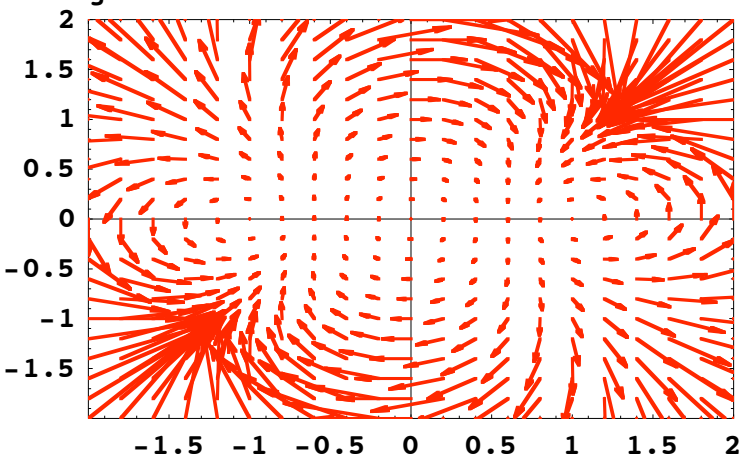


Fig 27. Φ: First integral of ODE

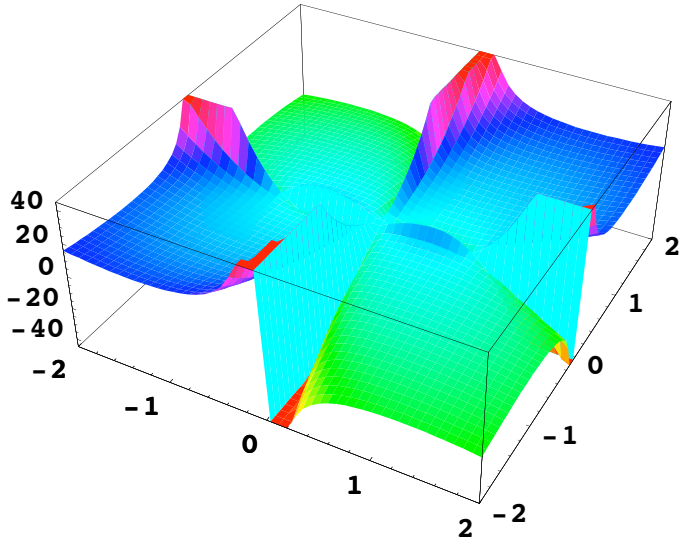
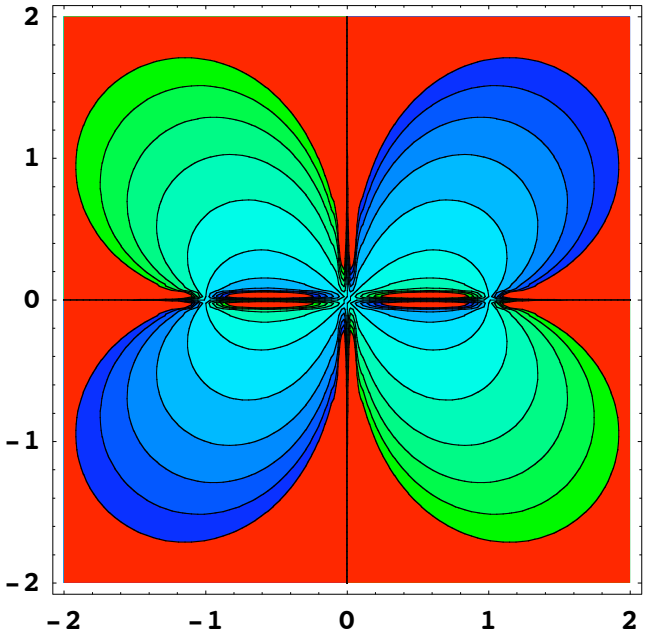


Fig 28. Φ = -10,-8,...,0,...,8,10



5. CONCLUSIONS AND FUTURE WORK

This report covers work performed during Year 2 and Year 3 of the project. A total of 8 PMs were spent by the Scientific Coordinator on Tasks S22 and S23 during these two years, addressing the topics discussed here and working alone. It has taken this long to realise that the disciplinary divisions are so deeply rooted between the domains of the project, and the methodological differences so marked even within the Science team, that it was impossible to build a constituency who could share the vision being advanced in this report. In hindsight more specialisation among the partners in mathematics and theoretical computer science would have been beneficial. On the other hand, the project also called for interdisciplinary integration, which at least at the level of the infrastructure was achieved fairly well.

Perhaps a greater investement in a team that could develop theoretical connections between computer science, mathematics, physics, and biology would have been wiser. European projects try to be friendly and collaborative endeavours. They place a high premium on building relationships that can last beyond any one project, which is a good thing for building research networks and for the continued building of a united Europe. Thus, where the Science partners did not share a common vision, priority was given to recognising the value that each partner brought to the table. Unification and integration of the theoretical vision, therefore, suffered in the interest of a greater overall constructive integration, and will take longer to be achieved.

If the 8 PMs spent to write this report, plus the 15.5 PMs of scientific coordination, have succeeded in highlighting the need for more concerted and concentrated research on what in this report are advanced as the core problems of biocomputing, then this relatively small effort will have been successful. In the OPAALS project a few of the DBE Science and Computing partners are continuing the effort begun here, with the help of a team of biologists. The LSE team is also being strengthened in the areas of mathematics, biology, and computer science in order to address the following topics:

π-Calculus and Concurrent systems
A better understanding of the modelling capabilities of π-Calculus (Milner, 1999; Sangiorgi and Walker, 2001) is needed. In particular, the concepts of strong and weak equivalence between labelled transition systems could be useful to augment the capability of search algorithms and the EvE fitness function by matching not just the service interfaces but also their internal processes.

Multiset rewriting, Stochastic π-Calculus, and Chemical programming
It seems worthwhile to develop a simulation facility that can be used equally well to simulate biochemical systems and the aggregation of sets of service components.

Fraglets
Fraglets will continue to be explored (also within the BIONETS project), extending them to include the functionality of DNA, and looking for ways to enforce symmetry constraints on their reaction behaviour. Because Fraglets were developed with process algebra, multiset rewriting, and engineering applications such as network protocols in mind, they seem like an excellent starting point for biocomputing. They also already support the specification of arbitrary topologies. If these topologies could be made variable at run-time, then the Fraglets would incorporate the full expressiveness of π-Calculus (currently they are closer to the simpler Calculus of Communicating Systems, or CCS, also developed by Milner (1999).

Network coding and Galois fields
Only recently did we realise that network coding uses finite groups as the starting point for a powerful theory of error detection and correction, which can be generalised to symbol systems of any alphabet size. (Moschoyiannis, 2001). These codes rely precisely on the inner structure and symmetry of Galois fields to reconstruct incomplete messages, suggesting the possibility for them to model an elementary form of gene expression.

Temporal Logic of Actions, BPEL, SBVR, and Transaction Models
TLA will be explored to model formally the behaviour of reactive systems (Kurki-Suonio, 2005) as it is also being used for the specification of the distributed transaction model being pursued by the UNIS Partner in DBE and OPAALS. This process view seems compatible with BPEL and SBVR and could therefore provide a link between the theory being pursued here and software engineering practice.

Update on the vision for automatic generation of code
The initial title of the workpackage "DBE UML Profile" assumed that biological behaviour could be integrated in the semi-formal language used by software engineers to specify the architecture of software applications and services. The new vision at the end of the research is to postpone the development of software engineering tools until the fundamental problem of mapping biological behaviour to computer science principles has been solved.

The intent remains to allow the high-level specification of functional and business requirements on the part of the users. This aim, in itself, is far from an easy problem, and it has also been addressed by the DBE project. Its solution however will not come from biology: a better understanding of natural language processing and logic will be required, as pursued by SBVR research. Rather than helping the process that we can generally refer to as formalisation of knowledge, the research initiated by this workpackage will hopefully improve the structural characteristics of SBVR and similar text-based or ontology based languages in order to facilitate the mapping from specification to running code.

APPENDIX *MATHEMATICA* PROGRAM FOR EFFECTIVE POTENTIAL ANIMATION

```
Clear[P, theta, s, thdot, U, x, y, z, w];
P = 1/10;
solution = NDSolve[{
    x'[t] == y[t], x[0] == 2/10,
    y'[t] == -(2 z[t] * w[t] / (1 + z[t]^2)) * y[t] - x[t]/(1 + z[t]^2), y[0] == 0,
    z'[t] == w[t], z[0] == 40/10,
    w'[t] == (y[t]^2 - P) * z[t], w[0] == 0},
   {x, y, z, w}, {t, 0, 300},
       Method → ExplicitRungeKutta,
       MaxSteps → 60000,
       PrecisionGoal → 6,
       WorkingPrecision → 12,
       MaxStepSize → 1/10];
```

```
Clear[P, theta, s, thdot, U, tmin, tmax, dt];
P = 0.1;
U[theta_, s_, thdot_] := (1/2) theta^2 - (1/2) s^2 thdot^2 + (1/2) P * s^2;
tmin = 0;
dt = 0.2;
numPts = 80;
Animate[
          thdot = Evaluate[y[tmax] /. solution][[1]];
          If[tmax/dt > 500, tmin = tmax - (300 * dt)];
          PlotCurveOnSurface3D[
                       {U[theta, s, thdot],
                            {theta, -1.5, 1.5}, {s, -4, 4},
                            PlotPoints → {40, 40}},
                       {{Evaluate[x[t] /. solution][[1]],
                         Evaluate[z[t] /. solution][[1]]},
                            {t, tmin, tmax},
                            PlotPoints → numPts,
                            PlotStyle → Thickness[.015]
                            },
                       {{Evaluate[x[t] /. solution][[1]],
                         Evaluate[z[t] /. solution][[1]]},
                            {t, tmax - .1, tmax},
                            PlotPoints → 2,
                            PlotStyle → {Thickness[.03], Hue[0]}
                            },
                       ViewPoint → {.3, -2.4, 3.},
                       Mesh → False,
                       Axes → False,
                       PlotRange → {{-4.1, 4.1}, {-4.1, 4.1}, {-1.4, 1.5}},
                       BoxRatios → {1, 1, 1},
                       Boxed → False  ],
      {tmax, .1, 300, dt} ];
```

REFERENCES

Alberts, B, Johnson, A, Lewis, J, Raff, M, Roberts, K and P Walter (2002). *Molecular Biology of the Cell*, 4th Edition, Garland Science, New York.

Armstrong, M A (1988). *Groups and Symmetry*, Undergraduate Texts in Mathematics, Springer. Preface.

Austin, J L (1962). *How to do Things with Words*, Oxford.

Babaoglou, O, Canright, G, Deutsch, A, Di Caro, G, Ducatelly, F, Gamgardella, L M, Ganguly, N, Jelasity, M, Montemanni, R, Montresor, A, and T Urnes (2006). "Design Patterns from Biology for Distributed Computing", *ACM Transactions on Autonomous and Adaptive Systems*, Vol 1, No 1, 26-66.

Banatre, J P, and D Le Metayer (1993). "Programming by Multiset Rewriting", *Communications of the ACM*, Vol 36, N 1.

Berdou, E and P Dini (2005). "Report on the socio-economics of Free/Open Source", DBE deliverable D18.3, sub-project 5, www.digital-ecosystem.org.

Bergé, P, Pomeau, Y, and C Vidal (1984). *Order within Chaos*, McGraw-Hill.

Briscoe, G (2002). "Biomimetics: Biological Design Patterns", MEng Thesis, Department of Computing, Imperial College London, www.doc.ic.ac.uk/~gwsb98 .

Chomsky, N (1957). *Syntactic Structures*, Mouton, The Hague.

Ciborra, C and O Hanseth (1998). "From tool to *Gestell*: Agendas for managing the information infrastructure, *Information Technology & People*.

Cuerno, R, Ranada, A F, and J J Ruiz-Lorenzo (1992). "Deterministic chaos in the elastic pendulum: A simple laboratory for nonlinear dynamics", *American Journal of Physics*, Vol 60, Number 1.

Cusumano, J P, and F C Moon (1995). "Chaotic non-planar vibrations of the thin elastica Parts I & II", *Journal of Sound and Vibration*, Volume 179, Number 2, 1995, pp. 185-208 and 209-226.

Davidovic, D M, Anicin, B A, Babovic, V M (1996). "The libration limits of the elastic pendulum", *American Journal of Physics*, Vol 64, No. 3, pp. 338-342.

Dawkins, R (1989). *The Selfish Gene*, 2nd Ed, Oxford.

Dini, P and E Berdou (2004). "Report on DBE-specific use cases", DBE deliverable D18.1, sub-project 5, www.digital-ecosystem.org.

Dini, P (2006). "DBE Science Vision", DBE Deliverable D4.1, www.digital-ecosystem.org.

Dini, P and F Nachira (Forthcoming, 2007). "The Paradigm of Structural Coupling in Digital Ecosystems", in *Toward Digital Business Ecosystems*, Corallo, A, Passiante, G and A Prencipe Eds., Edward Elgar.

Dini, P (2007). "A Scientific Foundation for Digital Ecosystems". In *Digital Ecosystems*, F Nachira (Ed), European Commission.

Drazin and Reid (1980). *Hydrodynamic Stability*, Cambridge.

Eigen, M and P Schuster (1977-78). "The Hypercycle: A Principle of Natural Self-Organization, Parts A, B, C", *Die NaturwissenShaften*, Vol 64, N 11, Vol 65, N 1, Vol 65, N 7.

Falk, L (1978). "Recurrence effect in the parametric spring pendulum", *American Journal of Physics*, Vol 46, Number 11.

Feenberg, A (2005). *Heidegger and Marcuse: The Catastrophe and Redemption of History*, Routledge.

Feynman, R (1965). *The Character of Physical Law*.

Feynman, R P, R B Leighton, and M Sands (1965) *The Feynman Lectures on Physics*, Reading, MA, Addison-Wesley.

Flores, F and C Spinosa (1998). 'Information technology and the institution of identity: reflections since 'Understanding Computers and Cognition'.' *Information Technology & People* 11, no. 4: 351-372.

Georgiu, I T (1999). "On the Global Geometric Structure of the Dynamics of the Elastic Pendulum", *Nonlinear Dynamics*, Vol 18, pp. 51–68.

Giavitto, J L, Malcolm, G and O Michel (2004). "Rewriting systems and the modelling of biological systems", *Comparative and Functional Genomics*, Wiley InterScience, Vol 5, 95-99.

Golding, D and P Wegner (2005). "The Church-Turing Thesis: Breaking the Myth", Computability in Europe (CiE) conference series.

Havey, M (2005). *Essential Business Process Modeling*, O'Reilly.

Hydon, P (200). *Symmetry Methods for Differential Equations: A Beginner's Guide*, Cambridge.

Kauffman, S (1993). *The Origins of Order: Self-Organisation and Selection in Evolution*, Oxford.
Kuhn, T (1996). *The Structure of Scientific Revolutions*, 3rd Ed, University of Chicago Press.
Kurki-Suonio, R (2005). *A Practical Theory of Reactive Systems*, Springer.
Kumar, S, and Bentley, P J (2003). "An introduction to computational development", in *On Growth, Form and Computers*, Kumar and Bentley Eds, Elsevier.
Maturana, H and F Varela (1980). *Autopoiesis and Cognition: The Realization of the Living*, Reidel, Boston. First published in 1973.
Maturana, H and F Varela (1998). *The Tree of Knowledge*: *The Biological Roots of Human Understanding*, revised edition, Shambhala, Boston.
Milner, R (1999). *Communicating and mobile systems: the π-calculus*, Cambridge.
Mingers, J (1995). *Self-Producing Systems: Implications and Applications of Autopoiesis*, Plenum Press.
Moore, J (1996). *The death of competition: leadership and strategy in the age of business ecosystems*, New York, Harperbusiness.
Moschoyiannis, S (2001). "Group Theory and Error Detecting/Correcting Codes", MSc Thesis, Department of Computing, University of Surrey.
Paun G and G Rozenberg (2002). "A guide to membrane computing", *Theoretical Computer Science*, Elsevier, Vol. 287, 73-100.
Priami, C (1995). "Stochastic π-calculus", *The Computer Journal*, 38(6):578–589.
Priami, C, Ed. (2005). *Transactions on Computational Systems Biology I*, Springer.
Olsson, M G (1976). "Why does a mass on a spring sometimes misbehave?", *American Journal of Physics*, Vol 54, Number 12.
Regev, A (2002). "Computational Systems Biology: A Calculus for Biomolecular Knowledge", PhD thesis, Tel Aviv University.
Regev, A, Panina, E M, Silverman, W, Cardelli, L, and E Shapiro (2004). "BioAmbients: an abstraction for biological compartments", *Theoretical Computer Science*, Elsevier, Vol. 325, N. 1, pp. 141-167.
Regev, A, and E Shapiro (2002). "Cells as Computation", *Nature*, Vol 419, 26 September 2002.
Rocha, L M (1998). "Selected Self-Organization and the Semiotics of Evolutionary Systems". In *Evolutionary Systems: Biological and Epistemological Perspectives on Selection and Self-Organization*. S. Salthe, G. Van de Vijver, and M. Delpos (eds.), Kluwer Academic Publishers, pp. 341-358.
Rocha, L M (2000). "Syntactic autonomy, cellular automata, and RNA editing: or why self-organization needs symbols to evolve and how it might evolve them". In: *Closure: EmergentOrganizations and Their Dynamics*. Chandler J.L.R. and G, Van de Vijver (Eds.) *Annals of the New York Academy of Sciences*. Vol. 901, pp 207-223.
Roux-Rouquié, M and M Soto (2005). "Virtualisation in Systems Biology: Metamodels and Modeling Languages for Semantic Data Integration", in *Transactions on Computational Systems Biology*, C Priami Ed., Springer.
Sangiorgi, D, and D Walker (2001). *The π-Calculus, A Theory of Mobile Processes*, Cambridge.
Searle, J R (1979). *Expression and Meaning*, Cambridge.
Stadler, B & P Stadler (2003). "Molecular replicator dynamics", *Adv. in Complex Systems*, World Scientific, Vol 6, 47–77.
Stephan-Otto Attolini, C (2005). "From Molecular Systems to Simple Cells: A study of the genotype-phenotype map", PhD Dissertation, Fakultät für Naturwissenschaften und Mathematik der Universität Wien.
Stanley, K O, and Mikkulainen, R (2003). "A Taxonomy for Artificial Embriogeny", Artificial Life, MIT Press, Vol 9, 93-130.
Thompson, D W (1992). *On Growth and Form*, Canto edition of original 1917 work, Cambridge.
Tondl, A, Ruijgrok, T, Verhulst, F, and R Nabergoj (2000). *Autoparametric Resonance in Mechanical Systems*, Cambridge.
Tschudin, C (2003). "Fraglets - a Metabolistic Execution Model for Communication Protocols". In *Proceeding of 2nd Annual Symposium on Autonomous Intelligent Networks and Systems (AINS)*, Menlo Park, USA, July 2003

Turing, A (1936). "On Computable Numbers, with an Application to the Entscheidungsproblem", *Proc. London Math. Soc.*, 42:2, 1936, pp. 230-265; A correction, ibid, 43, 1937, pp. 544-546.

Varela, F, Thomson, E, Rosch, E (1991). *The Embodied Mind: Cognitive Science and Human Experience*, MIT Press.

Winograd, T and F Flores (1987). *Understanding Computers and Cognition: A New Foundation for Design, Language and Being*, Addison-Wesley.

Wynskel, G (1993). *The Formal Semantics of Programming Languages*, MIT Press.

Yamamoto, L, and C Tschudin (2005). "Experiments on the Automatic Evolution of Protocols using Genetic Programming", Proc. 2nd Workshop on Autonomic Communication (WAC 2005), Berlin, Germany, October 2005.