Lecture notes on

Bigraphs: a Model for Mobile Agents

Robin Milner, November 2008

These notes are designed to be read in conjunction with the slides for six Lectures on bigraphs, with the above title. They aim to be useful for people teaching the subject to themselves or to others. The notes are numbered compatibly with the slides.

The Lectures are based on my book **The Space and Motion of Communicating Agents**, to be published by Cambridge University Press early in 2009 . They treat the topics in a different order, dictated by intuitions. The slides and notes can be studied without the book, but contain copious references to it (in footnotes, in order not to break the flow).

Introduction

Computing is transforming our environment. Indeed, the term 'computing' describes this transformation too narrowly, since traditionally it means little more than 'calculation'. Nowadays, artifacts that both calculate and communicate pervade our lives. It is better to describe this combination as 'informatics', which covers both the passive stuff (numbers, documents, ...) with which we compute, and also the activity of informing, or interacting, or communicating. The term 'ubiquitous computing' is used for a system with a population of interactive agents that manage some aspect of our environment.

These communicating artifacts will be everywhere. They will control driverless motorway traffic via sensors and effectors at the roadside and in vehicles; they will monitor and treat our health via devices installed in the human body and software in hospitals. The vision of ubiquitous computing is becoming real.

This realisation will make informatic behaviour into just one of the kinds of phenomena that impinge upon us, from a world in which we may no longer easily distinguish the natural from the artificial. Other kinds are physical, chemical, meteorological, biological, ..., and we have good understanding of them, thanks to an evolved scientific culture. But understanding still has to evolve for the behaviour of a population of informatic entities; we do not know how to dictate the appropriate concepts and principles once and for all, however well we understand the individual artifacts that make up the population.

These lectures do not aim to identify all the concepts involved in building models that will help everyone to understand the behaviour of ubiquitous systems. But the goal to understand them is just as compelling as the goal to understand (say) biological systems. The term 'everyone' here includes not only the informaticians who build such systems, but also the users embedded in them.

In these lectures I try to lay the basis for such conceptual understanding. It involves a low-level model for the structure and dynamics of such systems. This low level is utterly different from the low level at which we understand sequential computing. Typically a ubiquitous system, such as a body-area network that monitors and reports on a human's health, and may even administer medication, will comprise hundreds – or even millions – of mobile interactive agents, including sensors and effectors. The low-level model of such a system must represent the space in which they move and interact.

This mobile interaction is not only physical; it is also virtual. For such populations will include virtual—i.e. software—systems. These consist of software agents that move and interact not only in physical but also in virtual space; they include data structures, messages and a structured hierarchy of software modules. It appears that our low-level model must consist of a conflation of physical and virtual space, and therefore a combination of physical and virtual activity.

Models that can help to build and analyse such physico-virtual populations of agents will be as central to informatics in the 21st century as were the fundamental models of computing, by von Neumann and others, in the 20th. Forerunners of such models – for example process calculi – already exist, and these Lectures build upon them.

The argument for such a unified theory is detailed in the Prologue of my book **The Space and Motion of Communicating Agents**, to be published by Cambridge University Press early in 2009. The present Lectures are based upon the theory in that book. These notes consist of a commentary on the slides for the Lectures. The slides are at `www.cl.cam.ac.uk/~rm135/Bigraphs-Lectures.pdf`, and these notes are at `.../Bigraphs-Notes.pdf`. The notes are numbered compatibly with the slides, and are not meant to be read independently of them.

I shall modify the slides and the notes as time goes on. I shall therefore be grateful for comments and criticism of them, and of the book, sent to me at `rm135-at-cam.ac.uk`.

Robin Milner
Cambridge University, 2008

THE LECTURES

1. The first five lectures introduce bigraphs, with examples, as a self-managing structure suitable for ubiquitous systems. Some of the mathematics of bigraphs is developed, or at least represented graphically; this will allow a reader appreciate how the examples can be handled rigorously. It also prepares a willing reader for a detailed study of the book; with this in mind, I have provided many pointers to the book's definitions and theorems.

The sixth and final lecture outlines a strategy for modelling complex informatic systems hierarchically. Bigraphs are a candidate for lowest level in this tower of models; they are proposed as a **Ubiquitous Abstract Machine**. This term is meant to indicate that they not only constitute a low level of modelling, but also provide formal language for both specifying and programming ubiquitous systems. There is a close analogy with how the von Neumann machine has underpinned the analysis and programming of sequential systems.

I How agents are linked and placed independently

Bigraph structure

3. Here is a mixture of physical and virtual space. It shows how nodes can be linked, no matter 'where' they are, and also how a system may reconfigure itself.[1]

The top left large node may be Canada, and the lower right large node Australia. Sam (S) in Canada has sent a message M to Rachel (R) in Australia, which hasn't yet reached her. But M carries a key K which it inserts in a lock L, thus accessing a virtual agent A which will help M on its journey. This unlocking is represented by the 'reaction rule' shown at the bottom.

Meanwhile, R may move to China (top right), so M will have to chase her. Other reaction rules would represent R's move and the resulting chase.

[1] the Prologue

4. The next few slides are about structure only. They illustrate the 'bi' quality of a bigraph. The bare bigraph $\breve{G}$ has **nodes** $v_0, \ldots, v_5$ and **edges** $e_0, \ldots, e_2$. $\breve{G}$ is exactly represented by two constituents: a forest which is its placing (the node-nesting), and a hypergraph which is its linking (the edges). Conversely, every forest and hypergraph with the same node-set constitute a bare bigraph.[2]

This 'bi'-structure is vital, but we also want to build bigger bigraphs from smaller ones. So far, we call our bigraphs bare because they are not equipped with means to assemble them. To clothe a bigraph, we shall give it two **interfaces** or **faces**[3].

5. This picture highlights in red a set of nodes and edges that we want to consider to be a sub-bigraph $\breve{F}$ of $\breve{G}$. An interface will be needed to join it up with the rest of $\breve{G}$.

6. Here is $\breve{F}$, with its own (trivial) forest and its own hypergraph. Note that it has some links 'broken' where it was torn out of $\breve{G}$; also, its three nodes were torn out of different places in $\breve{G}$. We need to clothe a bare bigraph like $\breve{F}$ with linking and placing information.

7. For that purpose, we first define an interface $I = \langle m, X \rangle$ to have a **width** m and a **name-set** X. We clothe $\breve{F}$ by choosing $\langle 3, \{x, x'\} \rangle$ for its **outer** face and $\epsilon \stackrel{\text{def}}{=} \langle 0, \emptyset \rangle$ as its **inner** face. This yields a **bigraph** $F : \epsilon \to \langle 3, \{x, x'\} \rangle$. Width 3 means that F has three **roots** or **regions**[4], shown by dotted rectangles, while x and x' are the **outer names** of F. (We shall soon discuss inner faces).

Likewise we clothe $\breve{G}$ into a bigraph $G : \epsilon \to \langle 2, \emptyset \rangle$. Note that we have also made the forest and hypergraph of $\breve{G}$ into a **place graph** $G^{\mathsf{P}} : 0 \to 2$ and a **link graph** $G^{\mathsf{L}} : \emptyset \to \emptyset$ respectively. These are the **constituents** of the bigraph G.[5]

But we still have to describe the embedding of F into G.

[2] Chapter 1
[3] Definition 2.3
[4] Figure 1.2
[5] Definitions 2.1–2.3

8. When F is torn out of G it leaves a bigraph $H : \langle 3, \{x, x'\}\rangle \rightarrow \langle 2, \emptyset\rangle$. It has three **sites**[6] and **inner names** x and x'; thus its inner face is the outer face of F. We **compose** H with F at their common face, yielding $G = H \circ F$. Thus we are building bigraphs algebraically.

The bigraph H has its own constituents, the place graph H^{P} and link graph H^{L}. So we can also form G by composing place graphs and composing link graphs, and then combining the results.[7]

Built environment, Signature

9. For any application we need to define different **controls**, which are kinds of node. This is done by a **basic signature**[8] that also determines how many **ports** a node has. Here is the signature, and a typical state, for a built environment where movement and communication happen. The right-hand region of G consists of a room, but not the building that contains it.[9]

In blue is shown the algebraic expression for G. If K is a control with arity 3, say, then each K-node is written K_{xyz}, giving the names of the links impinging on it. A **closure** like $/x$ gives scope to a link name. This algebra will be defined later.[10]

10. The **contextual** bigraph H, composed with G, puts the right-hand room in a building. The two sites of H appear as id_1 in the algebraic expression.

11. Here is the composite, $H \circ G$. Note that the five A-agents are connected, perhaps in a conference call on their mobile phones. Also, each agent in a room is logged in to a computer, which is part of the local-area network for the building.

12. So far we have dealt only with static structure. For dynamics, there will be **reaction rules**[11] that change placing or linking or both. Here is a reaction that changes linking: one agent leaves the conference call.

13. Here is a reaction that changes placing: an agent enters a room —

14. — and this enables another linking reaction: the computer logs in the agent. Note that this linking reaction has a placing precondition: for logging in, the agent must be in the room.

15. Here are the reaction rules underlying these reactions. Consider rule (2); it is parametric[12]—the room may contain other occupants, e.g. other agents. It doesn't alter the linkage (if any) of the agent A.

Anatomy

16. Here are the important ingredients of a bigraph. Note the difference between an **open** link, which has a single outer name (e.g. y_0) and a **closed** link, also called an edge (e.g. e_0), which may link **points**—i.e. ports and inner names.[13]

II How to build complex systems from simple ones

Algebra

17. In this Lecture we develop the algebra of bigraphs. We continue to illustrate this for 'realistic' systems like the built environment; also, using CCS (Calculus of Communicating Systems) we illustrate how familiar models of processes can fit into bigraphs. We are still concerned only with static structure, not with dynamics.

The emphasis is on algebraic representation. One reason for this is the belief that a theoretically understood model should be the basis for designing a programming language, especially for complex applications such as ubiquitous systems, rather than that languages should be designed first and have theory retrofitted to them. Algebra is a powerful tool for

[6] Figure 1.2
[7] Definition 2.5
[8] Definition 1.1
[9] Example 1.2
[10] Chapter 3,Definition 3.2
[11] Example 8.1, Definition 8.5
[12] Definition 8.5
[13] Figure 1.2

modular construction, essential if complex programming is to be well-understood.

We also use mathematical constructions from **category** theory. We do not explore the deeper abstractions of categories, and we do not presume any previous knowledge of them. But we show in particular that the constructions of a **symmetric partial monoidal (spm)** category are a perfect basis for the more familiar operations of process calculi, and that they apply equally to a wider range of systems. Thus bigraphs become a framework for complex distributed systems.

A caveat for people who follow up the references to the book: As it explains[14], the book defines **concrete** bigraphs[15] first, and then **abstract** bigraphs in terms of them. In these Lectures we reach the intuitions sooner, though less rigorously, by defining the abstract ones first. We shall reach concrete bigraphs in Lecture V.

Elementary bigraphs

18. We begin with **elementary placings** and **linkings** [16]. These are the elements from which all node-free bigraphs can be built, using the categorical operations **composition** ($\circ$) and **tensor product** ($\otimes$) to be introduced shortly.

You can think of a placing either as a place graph whose interfaces are finite ordinals ($m, n, \ldots$), or as a bigraph whose interfaces are of the form $\langle m, \emptyset \rangle$—including an empty name-set. Similarly for linkings (mutatis mutandis).

The only other element, neither a placing nor a linking, is a **discrete ion**—a single node with distinctly named links for its ports[17]. The ion has a single site.

Basic operations

19. The notation $\langle P, L \rangle$ is convenient for combining a place graph and a link graph (with the same nodes and edges), just as we form an interface $\langle m, X \rangle$.

We 'define' **composition** here graphically, having seen examples of it in Lecture I. The formal definition[18] is important, because a secure theory cannot be based on pictures!

Just as we define composition of bigraphs in terms of composing their constituents, so we shall find later that many operations and properties on bigraphs can be defined separately on place graphs and link graphs and then combined. For example, a bigraph is an epi or a mono iff its constituents are epi and mono in their respective categories[19].

20. We now define how to place two bigraphs side-by-side to make a larger one, provided that their inner names—likewise their outer names—are disjoint. The juxtaposition is called **(tensor) product**; again, the formal definition[20] is important for rigour.

Bigraphs then form a **partial monoidal (pm)** category[21], a standard notion that places us in an understood theoretical frame. We are not *quite* standard, since juxtaposition requires disjoint names (as above), but the difference causes no difficulty.

21. Here, in pictorial form, are the equations that a pm category satisfies; they are easily verified for bigraphs. The one on the right is called the **bifunctorial** property of product ($\otimes$); this simply means that the order of product and composition can be inverted, allowing great flexibility in manipulation. With our graphical intuition, it is rather an obvious property.

22. Here we summarise what has been said above. Note that pm is a property of place graphs and link graphs just as it is of bigraphs.

23. We have not quite completed our categorical frame. We earlier defined the elementary bigraph $swap : 2 \to 2$ for swapping two places. it terms of this we can define $\gamma_{I,J} : I \otimes J \to J \otimes I$, called a **symmetry**; it swaps two adjacent *blocks* of places, those of I and J. This enriches our frame to become a **symmetric partial monoidal (spm)** category, pro-

[14] Introduction to Chapter 2
[15] Definition 2.3
[16] Definitions 3.1 and 3.2
[17] Definition 3.4
[18] Definition 2.5
[19] Definition 5.1, Proposition 5.2
[20] Definition 2.7
[21] Definition 2.10

vided that the symmetries satisfy the equations on this slide[22].

24. Here again we summarise what has been said—and again, it applies to place graphs and link graphs, as well as to bigraphs.

25. Placings and linkings are important for 'housekeeping' in bigraphs. It is nice that a node-free bigraph comprises one of each. We can blur the distinction between $\phi : X \to Y$, which is a link graph, and $\mathsf{id}_m \otimes \phi : \langle m, X\rangle \to \langle m, Y\rangle$ which is a bigraph with a trivial place graph constituent.

We shall now see a role for the node-free bigraphs; they will help us to derive familiar operations of process calculi.

Derived operations

26. Let us take a hint from process calculi. They often have a **parallel composition** operator, written $P \parallel Q$ or $P \mid Q$, where the processes P and Q may share named channels. Such processes have no inner face, but we can contrast this operator with our tensor product ($\otimes$) where sharing of names in the outer face is forbidden. The great advantage of the process operators is that they allow many processes to be combined, all sharing certain channels.

These operators—basic in process calculi—can be derived in bigraphs, using tensor product assisted by substitutions. We shall call them **parallel product** and **merge product**[23]. They have nice algebraic properties[24]. The definition given here is not so general as in the book, but is enough for the purpose of these Lectures.

Process calculi also have operators for **sequential composition**. In CCS this consists of prefixing an action μ to a process P, yielding $\mu.P$. Here this is generalised to a **nesting** operator[25], which has much wider application than for CCS. Like the derived products, the nesting $G.F$ allows F and G to share their outer names.

27. By defining these products on interfaces, we make it clear that the outer interfaces in derived product or nesting can share names. At the end of this Lecture, using these operators, we shall give a quite direct translation of CCS into bigraphs,

Sorting

28. Bigraphs over a given signature $\mathcal{K}$ allow arbitrary nesting of nodes, and arbitrary linking among ports. Often, in a particular application, some nestings and linkings do not make sense. So we may wish to reject certain interfaces and bigraphs from the category $\mathrm{BG}(\mathcal{K})$. This can be done by a **sorting** Σ. In a **place sorting**[26] we assign sorts to places, i.e. to roots, sites and nodes. **Link sorting**[27] does something analogous for linkage, and of course we can combine the two. In all cases, because we wish to work in an spm category, we insist that the formation rule of Σ does indeed confine us to a sub spm category—i.e. the formation rule is obeyed by identities and symmetries and preserved by both composition and tensor product.

29. As an example, we consider what may be a suitable place-sorting for the built environment[28]. We first introduce a sort for each control. In terms of these we impose the constraint that rooms can contain either agents or computers, but not other rooms or buildings. Similarly, buildings can contain rooms or agents. To ensure that the formation rule is preserved by composition we have to introduce disjunctive sorts like $\widehat{\mathsf{ar}}$.

Translating CCS

30. We now begin the representation of finite CCS in bigraphs; this example will run throughout the Lectures. Here is the usual CCS syntax[29]. Note that there are two syntax classes: *processes* denoted by $P, Q, \ldots$, and *choices* or *alternations* having the form $\mu_1.P_1 + \cdots + \mu_n.P_n$. (The case $n = 0$ is written 0.) This will be reflected in a sorting for bigraphs.

[22] Definition 2.11
[23] Definitions 3.11 and 3.15
[24] Propositions 3.12 and 3.16
[25] Definition 3.13
[26] Definition 5.1
[27] Definition 5.10
[28] Example 1.2
[29] Definition 5.3

We are not yet concerned with dynamics in bigraphs (this will come in Lecture III), but this slide recalls the essence of CCS reactions.

The **structural congruence** $\equiv$[30] relates expressions that intuitively represent the same process or alternation; the equations for $\equiv$ arise because linear syntax is unable to reflect this relationship. But bigraphs are a non-linear syntax; so we expect that, under our translation, structurally congruent expressions become identical bigraphs.

31. Here we see the sorting Σ_{ccs}[31]. It is an instance of an important class of sortings. The formation rule requires not only that the two sorts of node alternate in nesting, but also that whenever a root has sort θ, all its children (including sites) have sort θ. It is a good exercise to prove that the formation rule is thus indeed preserved by composition and product.

Although the bigraph for a translated CCS process is tedious to draw, it is not hard to understand; and the corresponding algebraic expression is reasonably terse. The null process is represented by the empty choice.

32. Finally, we give the translation[32]. It reveals many points of interest. First, a minor point: in translating the empty choice 0, X is used to denote the bigraph having only the idle names X.

Second, note how all CCS processes with free names $\subseteq X$ are translated to the same **homset** $\epsilon \to \langle 1 : \mathsf{pr}, X \rangle$. (In a category, a homset consists of all the arrows between two given objects.) Thus each CCS expression has an image in infinitely many homsets! This causes no difficulty, and there is a reason for it. Under a bigraphical reaction $g \longrightarrow\!\!\triangleright\, g'$, a link named x in g may well become empty in g'; but the formal treatment of reaction is simpler if the source and target of a reaction have identical interfaces.

Third, perhaps surprisingly, the merge product $|$ represents both parallel composition and alternation. Both are structural operations, but at different sorts. They are distinguished dynamically because of the bigraphical reaction rule for CCS, which operates at the sort of processes.

The theorem[33] represents the best possible static properties we could expect for the translation. Part (1) is achieved because the sorting formation rule has forbidden all bigraphs which do not represent a process. Part (2) indicates how structural congruence, when it was first conceived, foreshadowed the kind of non-linear syntax that is now realised by bigraphs.

The CCS running example is continued in Lecture III, where it will illustrate dynamic behaviour.

III Dynamical theory, illustrated for CCS

33. We have spent much effort on the structure of bigraphs. Its primary purpose is to support a simple but powerful dynamical theory. Treatments of dynamics abound in informatics. In bigraphs we are influenced by term-rewriting systems, by structured operational semantics of programming languages, by graph-rewriting, by Petri nets, and by process calculi. The book contains some description of these influences[34]. We have succeeded in one respect: we define only one notion of a reaction rule, which supports all our dynamical ramifications.

General formulation of reaction

34. First, as an example of existing process calculi, we recall the reactions of CCS as originally formulated. The single axiom in this slide shows how two processes in parallel, capable of respectively a positive and a negative action on the same channel x, can handshake on this channel and simultaneously discard their alternative actions. There are then two rules that declare reaction to be preserved by parallel composition and restriction, and a third rule declaring it to be preserved by structural congruence.

The important point is that reaction is preserved by all constructions except the prefixing of an action. Thus, in the axiom, reaction cannot occur within P or Q initially, but can do so after this initial reaction.

[30] Definition 5.4
[31] Definition 5.5
[32] Definition 5.6
[33] Theorem 5.7
[34] the Prologue, and Chapter 12

This reaction regime is specific to CCS; for example, CSP has a different regime. So it is not obvious how to generalise them for bigraphs.

35. Just as prefixing prevents reaction in CCS, so in a **bigraphical reactive system (BRS)**[35] we shall allow certain controls to prevent reaction, and others to permit it. We therefore further enrich a signature to become **dynamic**[36], by having it assign **active/passive** status to controls. This allows us say when a compositional context D preserves reaction.

In this definition we only consider **ground reaction rules**, i.e. those with a ground **redex** and **reactum**. This constraint will be relaxed on the next slide.

By equipping a BRS with an arbitrary set of rules, we allow a wide range of reactive disciplines. For example, any number of nodes can take part in a 'handshake' constituting a single reaction. This allows bigraphs to encode the communication discipline of CSP, where many agents may participate in an action on a single channel.

36. A **parametric reaction rule**[37], as here defined, is a way to generate a whole family of ground rules. There are many ways to define such families; this way is simple and powerful. The power lies in the **instance map** η; for it may be neither injective nor surjective, allowing the factors of the **parameter** d of the redex R to be discarded or replicated in forming the **instance** to be nested in the reactum R'. Thus there are two degrees of freedom in reaction; the reconfiguration of R into R' and the instantiation[38] of d into d'.

Parameters are required to be **discrete** [39] is for technical ease; this imposes no constraint, because any shared links can be created by the external context D of a reaction.

37. It is not surprising to find that reaction in CCS can be expressed by a single parametric rule as shown here[40]. Note that R and R' are **prime**[41], with a single root of sort pr. The parameter sites 0 and 2 of R have sort pr, while the sites 1 and 3 have sort ch.

The theorem[42] speaks for itself. Without it we could not claim to represent CCS in bigraphs.

Raw transitions and behavioural equivalence

38. For forty years or so it has been a vexed question what meaning is intended by a process expression. One thing is well accepted: if two processes expressions mean the same, then they should be interchangeable in any larger context—i.e. the meaning of an expression should not change when a subexpression is replaced by one with the same meaning. by the other. Algebraically, this property is expressed by saying that equivalence of meaning is a **congruence**.

The slide shows that 'having the same reactions' is not a congruence. What is lacking is what may be called a *conditional reaction*; we should characterize a process not only by the reactions it may perform without assistance, but also by the ways in which it may contribute to a reaction made with the assistance of its environment.

These conditional reactions are called **labelled transitions**[43], and were defined for CCS in 1980.

39. Here, the labelled transitions for CCS are defined in the same style as reactions were defined. The **labels** that describe (i.e. are witnesses of) conditional reactions are $\mu \in \{x, \overline{x}\}$, and the label τ means an unassisted reaction. The focal rule is the one for communication, which yields an unassisted reaction from two conditional ones.

Many different **behavioural congruences** can be defined in terms of these transitions.

40. A tractable and important behavioural congruence is **bisimilarity**[44], achieved by David Park in 1980-81 as a correction to Milner's work of 1980. It has a beautiful mathematical theory. It also yields

[35] Definition 8.6
[36] Definition 8.2
[37] Definition 8.5
[38] Definition 8.3
[39] Definition 3.8
[40] Example 8.1
[41] Definition 3.8
[42] Proposition 10.2
[43] Definition 7.8
[44] Definition 7.9

stronger and weaker behavioural congruences which are valuable for different purposes.

The notion is based upon defining what is *a* (not *the*) bisimulation. A prominent and elegant feature is that the union of any set of bisimulations is again a bisimulation; this is what entitles us to define bisimilarity as the largest one!

The proof of the theorem for CCS is not hard. But the labels μ are very specific to CCS, because it assumes that every reaction is a communication between just two participants.

41. To justify the general treatment of BRSs, we have to find a general form of behavioural witness—i.e. a form of transition label L that can be defined in all BRSs. This will lead in turn to a uniform behavioural theory. Of course, the bisimilarities for individual BRSs (e.g. other process calculi) will have specific properties; but to have them as instances of a common notion will help us to compare them.

Contextual transitions and behavioural equivalence

42. Let us call a transition or its label **raw**[45] if this label is defined without bigraphical means. Thus the CCS transition system is raw. By contrast, for a BRS we shall use **contextual** labels. The diagrams in this slide show the close relationship between reactions and contextual transitions. Both are based on a ground reaction rule; but a contextual transition has a commuting square involving the label L.

This created a sharp technical challenge that took years to resolve: How to limit the size of this square so that L is no larger than needed to complete a transition. The answer lies in an elegant concept from category theory.

43. Without answering this minimality question[46] in general—we shall do so in Lecture V—let us see what in means for CCS, as we have encoded it in bigraphs. The diagram shows a CCS agent a and a ground redex r, and you can see that they can overlap in different ways. How many ways?

44. One possibility is that a and r share their *left-hand* send-nodes. In that case or they also share their right-hand get-nodes, so the label L must supply the get-node of r, and (to make L minimal) that's all. To make the diagram commute, D has to supply the get-node of a.

45. Another possibility is that a shares its *right-hand* send-node with the left-hand get-node of r. Then it shares the rest of r too, so (for minimality) both L and R are node-free, but D has to adjust the linkage a bit.

Thus you can see that, given a and r, there can be different mini**mal** transitions, not a unique mini**mum** transition.

46. Having gained an intuition about minimal contextual transitions, we can now assert a completely general theorem about the corresponding bisimilarity[47].

In the course of reaching these results, it became obvious that the full structure of bigraphs was not needed to prove it. In fact, it holds for the much wider class of **wide reactive systems**[48], provided only that they possess a notion of minimal transition system. We shall prove this in Lecture V.

Compare raw and contextual equivalence for CCS

47. We must be sure that, in defining contextual transitions, we arrive at a behavioural theory that matches existing theories. We now do this for CCS. We discover a slight mismatch, revealing a feature of bigraphs that refines CCS. By ignoring this feature, we get a perfect match between the two notions of **raw** and **contextual** bisimilarity.

Raw transitions in CCS were defined inductively, with simple rules. But it is easy to characterize them syntactically (up to structural congruence)[49].

This is done in the table. Looking at case 1, we see that an $\overline{x}$ transition is possible for s if and only if, at top-level (i.e. not under a prefix), it has an alternation with a summand $\overline{x}.p$, and otherwise an arbitrary

[45] Definition 7.8
[46] Definition 7.13
[47] Corollary 8.8, Corollary 8.10
[48] Definition 7.2
[49] Figure 10.2

process q (which may ber empty) running in parallel. The components p and q may share restricted names Z, not including x itself. In case 3 we may have $x \in Z$, since a communication can occur on a channel whether restricted or not.

48. The same thing can be done for the minimal contextual transitions derived for CCS in bigraphs[50]. To obtain this characterization requires a detailed investigation, which we omit. Note that the derived labels are not complex. Corresponding to the raw label $\overline{x}$ in CCS, indicating that the process can 'output' on x, we have a label L which indicates that the context must 'input' on x for the communication to occur.

The Corollary[51] follows from our general theorem about bisimilarity for derived contextual transitions. On the other hand the Theorem[52] holds only if we drop case 4 of the contextual transitions. The reason is that case 4 allows the environment to perform a substitution on the agent g, and CCS has no raw label with this effect. In fact congruence of raw bisimilarity, as originally defined for CCS, does not imply that substitution preserves the equivalence.

Thus our results shed a new light on the existing theory, and reveal the need to make precise what congruence should mean.

IV Stochastic dynamics, e.g. for membrane budding

49. We can imagine that bigraphical systems such as our built environment can be made much more complex, and hence more realistic. The closer they claim to reflect a possible reality, the more we become concerned to simulate this reality closely, by somehow representing the probability of any given sequence of reactions. If this is done then a simulator may execute different sequences of actions each time it is run, but the collection of its runs will respect the relative probabilities.

For about a decade, stochastic process calculi have achieved this rather well. It is therefore natural to attempt it uniformly in bigraphs. These slides represent the obvious attempt, and illustrate it using ongoing work in modelling biological cells.

This also enables us to contrast the application of bigraphs to process calculi and to biology. Some differences appear, at least at first sight: process calculi are more concerned with behavioural equivalence, and are also concerned to admit links that cross place boundaries. On the other hand, biology seems more concerned with large populations of identical agents, e.g. protein molecules. This does not immediately suggest any variation in the definition of bigraphs. But the large difference of character between possible applications provides excellent opportunities to assess the bigraph model experimentally.

Rates of reactions

50. We wish to base our stochastic treatment entirely on attributing rates to reaction rules. This entails enriching each reaction rule by adding a strictly positive rate ρ to each rule. This is interpreted as the parameter of an exponential distribution of the execution time for reactions based upon the rule. As is well-known, given a set of possible reactions with different rates, whichever occurs first leaves the others (those remaining possible) with the same relative rates. Thus simulation, based upon continuous-time Markov chains, becomes tractable. This is usually expressed by saying that the exponential distibution is *memoryless*.

The rate of a given reaction $g \longrightarrow\!\!\triangleright\, g'$ depends not only on the underlying rule, but also on the number of distinct occurrences of the ground redex r in g, such that g' arises by substituting the ground reactum r' for r in g. Hence the importance of counting occurrences. This point will be discussed later.

The proposition is rather obvious, and depends on the fact that the rates on rules are strictly positive.

Membrane budding

51. We now examine a simplified model of a biological phenomenon. It exploits the placing of bigraphs in a way that is not available in the π-calculus.

Inside a cell, whose wall is a membrane, is a population of particles (shown in red). Outside are

[50] Figure 10.1
[51] Corollaries 10.3 and 10.4
[52] Theorem 10.6

protein molecules (shown in blue). Budding is induced when these coat proteins gather on the membrane. As the bulge increases, some of the red particles enter it. Fission occurs when enough proteins have gathered, creating a new cell. The process may be repeated.

The model has to deal realistically with the question: which (red) particles will belong to the new cell, and which will remain in the old one?

52. The diagram on this slide shows a bigraph representing a cell with two buds. The lower bud has separated, but is still linked to the coat proteins that caused its formation. The upper bud is under formation; so far only one coat protein linked to it. The presence of the two gates, connected across the membrane boundaries, means that fission has not yet occurred; they represent a virtual channel by means of which particles can migrate back and forth between cell and bud.

The place sorting constrains the place graph of the system. It resembles the place sorting for the built environment[53]. Note that the whole bigraph has a single region (root) of sort $\widehat{\mathsf{bc}}$, so it can contain only branes, buds and coat proteins.

53. There are essentially four reaction rules. In the first rule, the attachment of a single coat protein initiates a bud formation, creating the gated channel. The second rule allows more coat proteins to attach. The third rule allows particles to migrate along the channel in either direction. Economy is gained by using a redex with width 2; the rule does not have to mention a bud or brane. The final rule dictates that fission can occur when there are at least n coat proteins on the bud, where n is some fixed value.

Note that three of the rules are parametric; the parameter represents 'don't care' contents of a bud or brane.

We have not specifed rates for these rules. We imagine varying the rates in an computerised experiment, in the attempt to find out which rates best model reality. We may well wish to have two versions of particle migration, since the rates of migration into and out of the bud may differ.

54. We carried out one such experiment using the stochastic model-checker PRISM. We wanted to find how the number of particles in a separated bud would depend upon the rates of protein coating and particle migration. The slide shows how the experiment turned out.

Counting distinct reactions

55. When defining the rate of a reaction, we did not exactly define what was meant by a *distinct occurrence* of a redex. So let us illustrate what is meant, by an example that picks up the crucial points. We consider the possible reactions $g \longrightarrow\!\!\!\!\triangleright\, g'$ under different rules.

56. The first step in counting is to tag the nodes of g. There is only one B-node; so we just tag the A-nodes, using colours.

57. Now suppose there is just one rule, R; how many different occurrences are there of r in g?

58. In one way, r uses the blue node, and ...

59. ...in a second way , it uses the pink node. So there are two ways.

60. But if g has n A-nodes instead of two, there are n distinct occurrences or r in g.

61. Going back to g with two A-nodes, consider a different rule S.

62. There may seem to be two ways to match s in g, but in fact there is just one way; in s we have to colour the two A-nodes blue and pink, and there is only one way to do this because the members of a link are not ordered. To put it another way, the tagged bigraph s has an automorphism under bijection of tagging. Lecture V deals with tagged—or concrete—bigraphs, and will clarify this.

63. Finally, if instead g has n A-nodes, then there will be $n(n-1)/2$ ways, not $n(n-1)$ ways.

[53] Exercise 5.2

In summary, the way to get the count right is *first* to tag the nodes of the agent g, and *then* to count the ways to match the redex r in the tagged g.

Rates for transitions

64. Stochastic rates in a process calculus were pioneered over a decade ago. They were assigned directly to (labelled) transitions, because these were rightly regarded as more important than reactions for characterizing process behaviour. But in bigraphs, sometimes reactions are more important than transitions; in any case we have seen how to *derive* transitions from reaction rules, so we should also hope to derive rated transitions from rated rules.

Here, we see how to do it. It is rather close to how we derive rates for reactions, but we do it only for minimal transitions—so we are concerned only with minimal bounds. These are formally defined, via tagging, in Lecture V. So it appears that we may claim to have a simple and credible candidate for a uniform treatment of rates in bigraphs.

Experimental evidence is needed to confirm or refute this claim.

65. In CCS, since there is only one reaction rule we may feel a lack of freedom in defining the rate of a transition, which is a communication between two participants. We may like each participant in a communication to have the power to influence its rate.

The slide shows how this can be done to a considerable extent, without enriching the calculus. We have seen, in a biological example, how the rate of a reaction in a population of identical elements depends upon the size of the population. The slide shows how, by replicating any summand in a CCS alternation, we can multiple the rate of a communication by any integer. If large integers are used, there is little limitation in allowing only integer multipliers.

V Foundation for behavioural equivalence

66. In Lecture III we introduced **bigraphical reactive systems (BRSs)**[54], defining their reactions and transitions. We introduced the notions of minimal transitions and bisimilarity (illustrated using CCS) that form the basis on which, in an arbitrary BRS, one system can be understood to behave the same as another in every possible context. This equivalence depends on the external interactions of the system; we defined a minimal transition in terms of the minimal context needed to enable a system to perform a particular reaction.

We postponed until now the question of whether these minimal transitions exist. The purpose of this Lecture is to answer this question precisely. We shall do it uniformly, not merely for BRSs but for a much wider class of reactive systems of which they are an instance. We thus define a very general notion of behavioural equivalence, and thus a credible answer to the question: "What is a discrete process?"

67. We shall understand a process to be an equivalence class of descriptions of behaviour, e.g. the expressions in a process calculus, where two descriptions are equivalent if and only if they behave the same in some agreed sense. We want more than this: in a formalism that can build larger descriptions from smaller ones, we want the equivalence to be a **congruence**, in the sense that when two descriptions are equivalent we can replace one by the other in any larger description, without changing the latter's behaviour.

We want still more: the formalism must be able to encode in a natural way a wide range of existing process calculi, so as to provide a common theory for these. It must also possess a notion of space, enough impose spatial constraints upon behaviour.

Bigraphs satisfy the last criterion; but it turns out that there is a more general framework in which we can uniformly derive our behavioural congruence. To introduce it, we shall first define a general framework of **reactive systems**[55], too weak to express spatial constraints. We then enrich it just enough to express such constraints; thus we arrive at **wide reactive systems (WRSs)**[56]. The behavioural theory in this framework is very clear and clean, and can im-

[54] Definition 8.6

[55] Definition 7.1

[56] Definition 7.2

mediately be specialised to BRSs.

Discrete processes and tagging

68. We begin with examining the question "What is a discrete process?". The classical question "What is a computable function?" was solved by Alan Turing, giving the answer "a behavioural equivalence class of abstract machines". The meaning of "behaviour" in this case was in terms of input/output: the machine was said to compute a given mathematical function f if, given on its tape an argument x for f, it would in due course terminate with the value $F(x)$ on its tape.

An interactive process is less simple to describe than a computation. For one thing, it behaves non-deterministically, so in terms of input/output it must be said to compute a many-valued relation, not a function. But instead of delivering a single output (though different each time) for a single given input, it interacts continually with its environment. It is misleading to describe these interactions as either inputs or outputs; more accurately they are communications.

Let us at least confine ourselves to **discrete processes**, those whose behaviour is a sequence of atomic actions (e.g. communications) rather than a continuous activity. This still leaves open a wide choice of behavioural equivalences. Among those that have been studied in other settings we choose bisimilarity, though results analogous to ours can certainly be obtained for other equivalences and pre-orders on behaviour.

69. For this purpose the notion of **tagging**, to distinguish among the different occurrences of an agent within a system, becomes essential. We saw this in Lecture IV in determining reaction rates; we now see it in determining minimal transitions, and hence in defining **bisimilarity**[57]. In doing this, we shall work in wide reactive systems (WRSs), of which bigraphical reactive systems (BRSs) are a special instance.

The idea of tagging, labelling or identifying the occurrences of one entity in another is not prominent in algebra. It is used sometimes in computational calculi, for example in the λ-calculus—where it helps to prove theorems such as the (very central) Church-Rosser theorem. Even there, it has sufficed to treat it somewhat informally.

But here, we seem to need to treat it with full rigour. One can see the need in applications such as a built environment; we may wish to track an agent who moves about[58]. We have used it in Lecture IV to count the number of occurrences of a redex, and thus define the rates of reactions. And finally, in this Lecture we use it to define precisely what we mean by a minimal transition, which was so important in Lecture III.

70. A **concrete bigraph**[59] is just like an abstract one (which is what we have dealt with in the preceding Lectures), except in one respect: its nodes and edges have distinct identifiers, or tags. We denote the tags of G by $|G|$, called the **support**[60] of G. Thus, for example, if we were describing the journey of one particular copy of a broadcast message, as in the fanciful example[61] at the beginning of Lecture 1, we would use a tag to distinguish this copy from the others.

To treat this rigorously, we have to see how it affects the categorical framework we are using.

S-categories, reactive systems and transitions

71. Hitherto we have worked with **spm categories**[62]. We have to refine this, to accommodate support; the main reason is that composition and product must obey the discipline that tags within a single bigraph are unique. So for concrete bigraphs we need an **s-category**[63], in which these operators are only defined on bigraphs with disjoint supports.

This adjustment causes little difficulty. For example, it is independent of the notion of **sorting**[64]. Given a sorting Σ, we use $\grave{\mathrm{B}}\mathrm{G}(\Sigma)$—with a tagged name!—for the s-category of concrete bigraphs over

[57] Definition 7.9

[58] Section 11.1

[59] Definition 2.3

[60] Definition 2.4

[61] the Prologue

[62] Definition 2.11

[63] Definition 2.13

[64] Chapter 5

Σ, to distinguish it from the spm category $\mathrm{BG}(\Sigma)$ of **abstract bigraphs**[65].

Let us give more detail about the relationship between s-categories and spm categories, even though it is not essential for following the slides. In an s-category we say that g is a **support translation**[66] of f if it is obtained from f by a bijection $\rho : |f| \to |g|$ that respects the structure of f. Then we call the two arrows **support equivalent**[67], and write $f \simeq g$.

In the special case of two bigraphs F and G, we broaden this equivalence to **lean-support equivalence**[68], written $F \fallingdotseq G$, which also ignores idle edges (i.e. edges that are linked to nothing, which can arise from certain reactions). So an abstract bigraph A is just a lean-support equivalence class of concrete ones, such as G; we write $A = [\![G]\!]$.

72. We can define reaction for any s-category. Recall that there is an interface ϵ called the **origin**[69]; it is the unit for tensor product. A **ground** arrow is one out of the origin. So we can define reaction very simply, allowing it to happen anywhere (i.e. in any context D). What we cannot do is to limit *where* it can happen, because there is no sufficient notion of *place* in s-categories. To be more precise, the symmetries provide an elementary notion of swapping places, but an arbitrary arrow provides no structuring of places.

73. As promised, in a **wide reactive system** we add just enough to represent how arbitrary arrows can structure places, without going as far as the nodal structure of bigraphs. First note that there is a very simple s-category NAT, whose objects are natural numbers considered as ordinals $m = \{0, \ldots, m-1\}$, and whose arrows are functions $f : m \to n$. For $i \in m$, if $f(i) = j \in n$ then we can think of this as saying "place $i \in m$ lies via f in place $j \in n$". To enrich an arbitrary s-category **`C** with this kind of placing, we just insist that there is a **functor**[70] 'width' from **`C** to **Nat**. A functor of s-categories, just as in spm categories[71], is required to satisfy various conditions. Thus, when we form a complex arrow $f : I \to J$ in **`C** from simpler ones, we can derive how f relates places in I to places in J from the placing structure of the simpler arrows.

In terms of this placing structure, the slide then shows how to equip a reactive system with an **activity relation** Act[72] which tells us, for every arrow $f : I \to J$, whether or not it is *active* at each place in its inner face I. Thus, by supplying width and Act, we declare a reactive system to be wide. It turns out, of course, that BRSs are wide (and have a lot more structure too).

74. A key property of the width functor is that, given $f : I \to J$, it determines a subset $\mathsf{width}(f)(\mathsf{width}(I))$ of $\mathsf{width}(J)$ which is a location $\tilde{j} \subseteq J$. It is the image of I in J under width function of f.

So we can refine our reactions and transitions to become wide ones. First we require the the context D of the redex $r : I \to J$ is *active*, i.e. active everywhere in $\mathsf{width}(I)$. Second, we index the reactions and transitions by the image of I under the width function of D. This latter condition is needed to ensure congruence of bisimilarity.

75. Having dealt with width—i.e. *where* transitions can happen—we now look at *how* they happen. This entails understanding how an agent a can overlap with a redex r. This overlap is just their support intersection $|a| \cap |r|$.

For a transition to be *minlmal* we need to identify the part of r that doesn't overlap with a. Our diagram looks persuasive, but is it uniquely defined?

Relative pushouts

76. To determine the meaning of a minimal triple, as was sought on the previous slide, we need to look at a basic categorical phenomenon, which is a refinement of the classical notion of a **pushout**[73].

Some basic terminology: a **span**[74] is a pair of $\vec{f} = (f_0, f_1)$ of arrows with the same domain (= in-

[65] Definition 2.19
[66] Definitions 2.13 and A.1
[67] Definition 2.13
[68] Definition 2.19
[69] Definition 2.10
[70] Definition 2.9
[71] Definition 2.11
[72] Definition 7.2
[73] Definition 4.2
[74] Chapter 4

ner face), and a **cospan** $\vec{g}$ is similar but with the same codomain (= outer face). Then $\vec{g}$ is a **bound**[75] for $\vec{f}$ if $g_0 \circ f_0 = g_1 \circ f_1$.

If so, maybe $\vec{f}$ has a 'smaller' bound than $\vec{g}$. This would take the form of a triple $(\vec{h}, h)$ as stated in the first diagram, called a **relative bound** for $\vec{f}$ relative to $\vec{g}$. (h measures how the bound has shrunk.) We want a relative bound that is as small as possible; then we shall call it a **relative pushout (RPO)**[76]. It has the property that it is at least as 'small' as any other relative bound $(\vec{k}, k)$, in the sense of the second diagram.

RPOs don't always exist; in fact they exist neither in abstract link graphs nor in abstract bigraphs[77]. But they dol exist in *concrete* bigraphs[78].

77. Intuitively, if the bound created by an RPO is minimal, then any attempt to decrease it further should be vacuous! This motivates the definition of **idem pushout (IPO)**[79]; it is simply a bound which, with an identity as third member, constitutes an RPO for itself. There follows a list of beautiful properties of RPOs and IPOs[80]. In particular, the second and third properties represent the intimate relationship between IPOs and RPOs.

The fourth property (in two parts) is exactly what is needed in the forthcoming proof of behavioural congruence. The reader familiar with pushouts[81] will recall that these two properties, the cutting and pasting of IPOs, also belong to pushouts. Indeed, they may lead us to expect an IPO to be exactly a pushout! But it is not: the key difference is that for a given span there is typically a family of IPOs; on the other hand any pushout is unique up to isomorphism. So it is accurate to call an IPO a mini**mal** bound, and a pushout a mini**mum** bound.

RPOs (hence IPOs) exist in concrete bigraphs[82] but typically not in abstract bigraphs[83]. They constitute a pleasant example of the combining place graphs and link graphs[84]; to get an RPO in bigraphs we get RPOs for their constituent place graphs and link graphs, and then combine them. In developing the theory if bigraphs, this separation woas found to simplify the task by an order of magnitude!

78. This and the following slides illustrate the RPO construction for link graphs. We begin with a simple link graph G.

79. We decompose G into $D_0 \circ A_0$, choosing A_0 to contain the nodes v_0, v_2, v_2' and the edge e_0 ...

80. ...and again decompose G into $D_1 \circ A_1$, choosing A_1 to contain the nodes v_1, v_1', v_2, v_2' and the edge e_2. Thus the span $\vec{A}$ is bounded by the cospan $\vec{D}$.

81. We then decapitate D_0 and D_1, removing their shared (upper) part...

82. ...leaving the RPO $(\vec{B}, B)$ for $\vec{A}$ relative to $\vec{D}$.

This example gives good intuition on RPOs, but the formal expression of the construction[85] needs care. A detailed case analysis is needed to prove the construction is sound[86], that it yield a relative bound[87] and that this bound is indeed an RPO[88].

The RPO construction for a place graph RPO is remarkably similar, and the combination into a bigraph RPO is straightforward.

Minimal transitions and congruent bisimilarity

83. We now declare that a *minimal* transition is one based upon an RPO. We have also explained the need for *wide* transitions, i.e. transitions indexed by a location. The next step is to adapt our definition of bisimilarity[89]. As we are in concrete bigraphs, when defining a bisimulation containing the pair (a, b) we

[75] Definition 4.1
[76] Definition 4.3
[77] Example 5.25 and Figure 5.4
[78] Section 5.1
[79] Definition 4.4
[80] Proposition 4.5
[81] Definition 4.2
[82] Section 5.1
[83] Exercise 5.25
[84] Theorem 5.11
[85] Construction 5.5
[86] Lemmas 5.6
[87] Lemma 5.7
[88] Theorem 5.8
[89] Definition 7.9

are only concerned with labels L such that $|L|$ is disjoint from $|a|$ and $|b|$. This is the only adaptation needed.

84. The proof that bisimilarity is a congruence in a concrete WRS with RPOs is remarkably elegant. It replaces some proofs for particular process calculi that are long and tedious. The proof is outlined here, emphasizing the vital role played by cutting and pasting IPOs. Some details involving the handling of support equivalence[90] have been omitted.

This proof is, of course, valid for concrete BRSs, since they possess RPOs.

85. However, a process calculus (such as CCS) is typically encoded as an abstract BRS, not a concrete one. So, in order to complete the correspondence of bigraphical theory with known process theories, we must transfer the above results to abstract BRSs.

This slide outlines the procedure, which is detailed in the book[91]. Briefly: we start with an abstract BRS; we tag everything, making it concrete; we get our minimal transition system and bisimilarity congruence in the concrete BRS; finally we carry them back to the abstract one. Of course, it has to be proved that they survive the journey!

VI Ubiquitous Systems: a Context for Bigraphs

Models and their tower

87. Ubiquitous systems may be very large; to understand them, we need **models** at many levels. At a high level, a model may use sophisticated – even quasi-human – properties to describe the behaviour of a system. For example, one such concept is **reflectivity**; it represents the ability to analyse one's own behaviour.

A **tower** of models will be both high and wide. Higher models will be **abstractions** or **explanations** of lower ones; at the lowest level, a model explains a real physical system. The tower also has width; models of different subsystems of a system may be **combined**, i.e. juxtaposed but with some elements in common. Thus explanations of the subsystems can be combined into an explanation of the whole.

In one sense we can regard realities, i.e. real physical systems, as extremal models. They are extremal because they explain nothing else. But, as we see later, the explanations of realities have different scientific status from explanations of models.

We proceed to make these ideas more definite with the help of examples.

The nature of models

88. What is a model? At very least, it contains a set of **entities**. A programming language P is a model, whose entities are programs and their syntactic components. In what sense does it explain a reality? Take the example of an initialised computer C; we may say that P explains C, or equivalently that it C **realises** P, in the same sense that the abstractions of physics, such as electrons and quarks, explain physical realities. In our case, the difference is that a computer is an *artificial* reality; and it often comes into existence after the model which explains it. For this reason, we may replace the word 'explain' by 'specify'.

89. But we cannot say that P explains C unless we declare that the entities of P have **behaviour**, defined abstractly. We therefore refine our definition of a model to include the behaviour of its entities.

Then, to say that C realises P, we require a specific relationship between P's defined behaviour and the **observations** we make of C's behaviour. These observations consist of keyboard and screen events. For example, for the program 'x:= 3 + 5; print x' we expect the keyboard events that realise the program are followed by a screen event displaying '8'.

In this way, we have assimilated the way programs are realised by artifacts (computers) to the way in which a scientific model explains natural phenomena. Of course this assimilation exists not only for informatics but for any engineering discipline and its artifacts.

Composing explanations

[90] Definition 7.13
[91] Section 7.4

90. We have describes how a programs specifies (or explains) an initialised computer. This explanation can be unpacked into several explanations via intervening models – each with their special notion of entities and behaviour. And programs (say in **C** or Java) may themselves be explained by logical specifications. The logic's entities are not dynamic; they are logical formulae, and their behaviour is better called their **meaning**. Logicians call it their 'valuation'; it gives each formula a truth-value for every possible assignment of values to its variables.

Thus our diagram concatenates, or **composes**, four explanations. At the top level, the explanation may be by means of Hoare's logic; to each program P is assigned a set of sentences of the form $\{F\}P\{F'\}$, where the formulae F and F' may be called pre- and post-conditions. The explanation is validated by showing that whenever F is true at the start, and P runs to termination according to its rules of behaviour, then F' will be true at the end. As everyone knows, this valuation can be formalised and proven (or disproven) by Hoare logic.

The explanation at the next level down is a compiler; its validation is sometimes called a proof of compiler correctness.

Lower still, assembly code programs are modelled by hardware design diagrams. This design can be formally described and the validation can be done formally; indeed, this has been done by automatic reasoning for simple computers or their parts.

The lowest level consists of hardware diagrams that specify real computers. This is a specification of a reality, so its validation can only take the form of observations of expected behaviour. Thus our original explanation has been factored – and extended – into three explanations that can be formally validated, and one that can only consist of predicted observations. We shall come back to this distinction.

Mathematicians may well regard an explanation as a commuting square; but we shall avoid mathematical terminology, because we don't wish to constrain our models and explanations to be rigorous. Some of the best and most-used explanations are done in natural language; an outstanding example is the Algol60 Report.

Combining explanations

91. By composing explanations we can build a tower high. Towers also have width, arising from juxtaposing or **combining** models. The example here shows a complete model of an aircraft, achieve by combining an informatic model of its embedded software with an electro-mechanical model of its other engineered parts. The latter model could itself be a combination of an electrical and a mechanical model. Such combinations typically involve shared behaviour; for example, certain electrical events and software events are shared. One can then regard the complete explanation as combined from the partial explanations. It is likely that, to be tractable, a complete explanation must be factored in this way.

92. To model the flight of a complete aircraft, it is not enough just to model the artifact. A further combination must be made, this time with a meteorological model – a model not of an artifact but of a natural reality. It is only because we have assimilated these two kinds of model that we can treat their combination. And again, certain events – such as the impact of a gust of wind – are shared between them.

93. Each of the three models we have combined may, of course, be decomposed vertically; equally, it may be extended upwards by further more abstract models. For the informatic model, a fine example of this is the recent analysis – i.e. explanation – of the embedded software, using particular techniques of abstract interpretation, carried out for the European Airbus by a French team at INRIA.

We can note two features of this impressive success. First, several distinct aspects of the software were explained independently, and this rendered the combined explanation tractable; it was indeed carried out with automatic assistance. Second, the total explanation was a special case of combination, which did not rely upon events shared between the partial explanations.

The unifying influence of model explanations

94. This slide summarises the general approach to models that we have illustrated. We should not claim

to have defined models rigorously; there may be many kinds of model, and some compositions or combinations may not make sense. But, having seen examples, we must agree that both composition and combination are essential members of our informatic toolkit.

There is a welcome initiative of Model-Driven Engineering (MDE) in the software engineering profession. What is sometimes missing there is the explicit presentation of the *behaviour* of entities. Sometimes this is justified in the object-oriented approach, because the behaviour can be assumed to be in terms of objects. Our examples show that this assumption is far from justified in general; there are many ways to define behaviour.

95. Having assimilated the modelling of artifacts with modelling in natural science, we have to accept that we can never formally ascertain the behaviour of real informatic entities with certainty. This has been pointed out before, in particular by Michael Jackson.

Note how this applies to the combination of models. For example, in combining a model of embedded software with an electro-mechanical model of an aircraft we must identify the events shared between the models. We may believe we have identified them all; but this cannot be formally verified. For instance, we may wrongly neglect the effect of heat exchange between an embedded computer and its mechanical environment.

The sheer size of software entities forms a huge barrier to understanding them, and we have no choice but to decompose or factorise large explanations into smaller ones. So it is essential to aspire to rigorous models, and (except at the very lowest level) to rigorous validations between them. This aspiration is a large part of the claim that informatics has scientific status; not only as an experimental science, but also as an analytical science in the sense of applied mathematics or logic. Indeed its domain of application will increasingly pervade our world.

96. Though we aspire to science, we are also an engineering discipline and a production industry. For much of our activity, modelling and explanation are informal. To achieve a formal specification we need first to build informally specified artifacts and play with them.

A related point is that no model suits everyone; a simple model – even though incomplete – may have great value for non-expert users and executives. A good example is *message sequence charts*; they are designed for non-experts, but although they only describe finite fragments of behaviour, they often reveal essential points for design. They can be seen as partially explaining a more rigorous model, perhaps a concurrent programming language.

But some informal models never achieve formal definition. This is true even of programming languages, which are the platform for large systems upon which we increasingly depend. Their informal definitions may be ambiguous, leading to implementations that are inconsistent or unintended.

As informatic systems become more complex and autonomous, can we afford not to refine our their informal specifications into formal ones? This is the only way in which we shall truly understand such a system, either before or after we build it.

At least part of our aspiration to science must be to define models from which programming languages are unambiguously derived.

Ubiquitous systems: qualities and concepts

97. The two quoted visions of ubiquitous systems agree on their most prominent quality: they will support us without our awareness. Since we are unaware of them (unless we are experts) we shall not understand them. Then how do we, as a society, have confidence in these systems?

Other engineering disciplines have grown at a relatively slow pace, and society has a stratified understanding built upon a hierarchy of expertise. For example, there are many levels of understanding of electrical systems, from the electrical scientist through the acedemic epelctrical engineer, through levels of understanding in the construction industry, out to the electrician who installs home systems.

The pace of growth in the informatic discipline has been hectic by comparison; society therefore lacks a hierarchical understanding. In the case of ubiquitous systems the lack presents a real danger. There

is hope for this to be recognised, since the difference from previous software systems is so great; there is a clear incentive to build a hierarchy of expertise in parallel with building the first generation of ubiquitous systems.

98. A ubiquitous system is startlingly different from traditional computing. It consists typically of a population of agents, both hardware (sensors and effectors) and software, interacting and moving, with behaviour that is not fully determined. It (and its agents) will take decisions autonomously; often the agents will **negotiate** with each other, will **trust** or **mistrust** each other, and will **know** something about the system as a whole. They will be larger than any systems we know, will interact and perhaps merge with each other, and adapt in other ways to their changing environment.

A simple example of interaction will be between a system controlling (and driving) traffic on the motorways, and a health care system managing patients' well-being in their homes. If I have an emergency then the health system will send an ambulance, which will have to negotiate with the traffic system for a fast passage on the motoway.

99. A huge range of concepts will be involved in understanding such systems, and hence in specifying their behaviour. The list in this slide is certainly not exhaustive. No single model can handle all these concepts; some tower of models is essential, in which a higher-level concept such as trust will be reporesented by more concrete behaviour lower down. For example, 'A trusts B' may be implemented as 'A has a log of previous successful interactions with B'.

Note too that properties related to safety and security are prominent.

100. In seeking to lay the foundation stones of this model tower, i.e. the lowest level of modelling, we look for very concrete properties. It is natural to choose properties of **structure** and **motion** – the latter being the way that structure varies with time. Structure is concerned with **connectivity** and relative locality of agents. Finally, given that motion is likely to be non-deterministic, we add **stochastics** to our list of ground-level concepts for ubiquitous systems.

Let us use the term **abstract machine** to denote the way these concepts are related; this machine, specialised for each ubiquitous system, will describe possible behaviours of that system, including its internal behaviour and the way it interacts with its environment. Call it the **Ubiquitous Abstract Machine (UAM)**. It differs strikingly from the von Neumann machine, which has served for half a century as the abstract mechanism underlying sequential computing. The success of the UAM will be the extent to which it provides a foundational model that can implement the higher models required for ubiquitous systems.

This is how the bigraph model arose. It is a development of the family of process calculi that have increasingly refined the way in which interaction depends upon both **placing** (locality) amd **linking** (connectivity). The model is only a proposal; it can only become foundational model for ubiquitous computing if it survives serious experimental application. For the latter, it must be seen to yield language for programming and simulation, and equipped with appropriate mechanised tools for analysis, such as model-checking.